Scheduling Optimization for Resource-Intensive Web Requests on Server Clusters

Huican Zhu, Ben Smith, Tao Yang
Department of Computer Science
University of California
Santa Barbara, CA 93106
{hczhu, besmith, tyang}@cs.ucsb.edu

Abstract

Clustering support with a single-system image for large-scale Web servers is important to improve the system scalability in processing a large number of concurrent requests from Internet, especially when those requests involve resource-intensive dynamic content generation. This paper proposes scheduling optimization for a Web server cluster with a master/slave architecture which separates static and dynamic content processing. Our experimental results show that the proposed optimization using reservation-based scheduling can produce up to a 68% performance improvement.

1 Introduction

There are two types of information serviced at a Web site: static data, whose service is implemented as a simple file fetch, and dynamic data, which involves information construction at the server before users' requests are answered. Recently more Web sites generate dynamic content because it enables many new services such as electronic commerce, database searching, personalized information presentation, and scientific/engineering computing [7, 23]. Since dynamic content generation places greater I/O and CPU demands on the server, the server bottleneck becomes more critical compared to the network bottleneck and it limits the scalability of such servers in processing large numbers of simultaneous client requests. Examples of such systems can be found in IBM's Atlanta Olympics Web server and the Alexandria Digital Library system [3, 4, 19].

Clustering with a single-system image is the most commonly used approach to increase the throughput of a Web site, motivated by the recent advance in clustering computing research [2]. With a server cluster, multiple servers behave as a single host from the clients' perspective. NCSA [22] first proposed a clustering technique that uses DNS rotation to balance load among cluster nodes. Research has demonstrated that DNS round-robin rotation does not evenly distribute the load among servers, due to non-uniform resource demands of requests and DNS entry caching. A number of projects [5, 9, 19] have proposed methods for more fairly distributing load among a group of servers based on HTTP redirection or intelligent DNS rotation. The main weakness of a DNS-based server cluster is that the IP addresses of all nodes in a cluster are exposed to the Internet. When one server goes down for administrative reason, or due to machine failure, a client may still attempt to access this machine because the DNS server is unaware of the server's status, or the IP address of this machine is cached at the client's site. Hiding server failures is critical, especially when global customers depend on a Web site to get vital updated news and information. To address this issue, one solution is to maintain a hot-standby machine for each host [31]. Another technique for clustering Web servers with load balancing and fault tolerance support is load balancing switching products from Cisco, Foundry Networks and F5Labs [8, 14, 13]. Switches use simple load balancing schemes which may not be sufficient for resource-intensive dynamic content. Neither DNS nor switch based solutions provide a convenient way to dynamically recruit idle resources in handling peak load.

Recently a master/slave architecture [15, 35] has been proposed for clustering Web servers. Such an architecture has advantages in dynamic resource recruitment and fail-over management when integrated with switching or hot-standby techniques, and it can also improve server performance compared to a flat architecture. Instances of M/S architecture can be found in current industry Web sites such as Web searching sites at Inktomi [18] and AltaVista [12]. A master/slave architecture organizes server nodes in two levels. The master level accepts and processes both dynamic and static content requests. The slave level is only used to process dynamic contents upon masters' requests. As presented in [35], this architecture can be made efficient by separating static and dynamic content processing, low-overhead remote execution of CGI requests, and reservation-based scheduling which considers both I/O and CPU utilization. We have shown that these schemes can effectively achieve load re-balancing and the remote execution overhead is not only negligible but even smaller than standard local CGI execution. We

To appear in the Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'99),Jun. 1999

do not use HTTP redirection [5] for request re-scheduling because it adds client round-trip latency for every rescheduled request and also exposes IP addresses of server nodes.

The work on a layered architecture for network services in [15] does not have a detailed study on performance optimization and evaluation for issues specific to Web servers. This paper presents a theoretical study and heuristic design for request scheduling on a Web server cluster with a master/slave architecture. We provide extensive evaluation using trace-driven simulation and performance measurements to demonstrate the effectiveness of the proposed scheduling optimization. The rest of this paper is organized as follows: Section 2 gives a background overview of Web clustering and describes the M/S architecture. Section 3 provides a theoretical analysis to give insights and conditions when the M/S architecture can outperform a flat architecture. Section 4 presents the scheduling heuristic based on our theoretical result. Section 5 describes the experimental results. Section 6 discusses related work and concludes this paper.

2 Backgrounds and the Master/Slave Architecture

In the World Wide Web environment, clients access an Internet server using the Hypertext Transfer Protocol (HTTP). To create dynamic content in response to an HTTP request, most servers implement the Common Gateway Interface (CGI) [29]. The CGI interface requires that the Web server initialize an environment containing the script parameters, and then fork and execute the script. As a result, every CGI request requires the creation of a new process.

Web servers with a single-system image can be clustered using DNS rotation in which a domain name server (DNS) cyclically maps the IP addresses of a Web site to clustered nodes [5, 22]. In such a setting, the client determines the host name from the URL, and uses the local Domain Name System (DNS) server to resolve its IP address. The local DNS may not know the IP address of the destination, and may need to contact the DNS system on the destination side to complete the resolution. After receiving the IP address, the client then sets up a TCP/IP connection to a well-known port on the server where the HTTP process is listening. The request is then passed in through the connection. After parsing the request, the server sends back a numeric response code followed by the results of the query. The connection is then closed by either the client or the server.

One problem with this DNS-based approach is that load imbalance may be caused by client-site IP address caching. Moreover, a client site cannot be aware if a machine is no longer in service and may be denied service if that client tries to use a cached IP address to access a dead node. A solution for masking server failures is called "failover" using hot-standby techniques [31]. This solution dictates that one computer be used to monitor another one and it is activated to take over if the monitored computer crashes. Industry has sold such a solution for many years and the cost of such a solution is expensive since the machine that monitors is not utilized if there is no failure. Another solution is to use recently developed load balancing switching products from Cisco, Foundry Networks, and F5Labs. A switch can cluster a logical community of servers represented by a single logical IP address. These products also provide load balancing schemes to distribute application traffic to server nodes. Switches also provide sub-second failure detection to eliminate a dead node from the server pool. This ensures that traffic continues to flow and services are still available for processing new client requests.

Clustering based on DNS rotation or load balancing switches still cannot deliver satisfactory performance for Web sites with intensive dynamic content processing for the following reasons: 1) Adding nodes to the DNS list or to a load balancing switch requires a manual change of system configuration, which is not convenient and efficient for recruiting non-dedicated computer resources [1, 2]. 2) The capability of load balancing provided in a switch is still limited because a switch must forward packets as fast possible. 3) The class of dynamic content requests normally require much more computing and I/O resources, compared to static file retrieval requests. Mixing static and dynamic content processing can slow down simple static request processing.

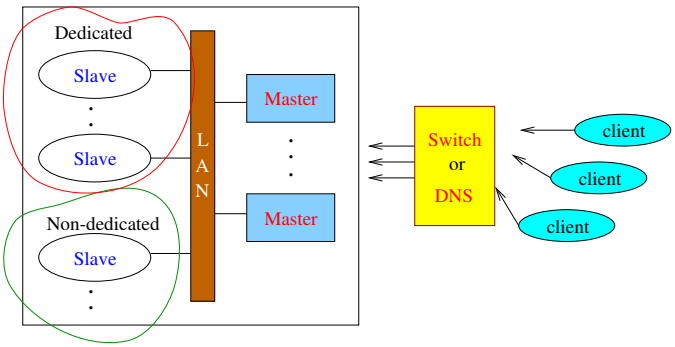


Figure 1: The master/slave architecture.

This paper uses term "flat architecture" to refer to the previous DNS or switch based approach in general. Recently the master/slave architecture (M/S for short) with better efficiency, expandability and availability has been proposed [35] based on the layered network service research [15] and the design of Inktomi and AltaVista searching engines. As depicted in Figure 1, an M/S architecture contains two levels. Master nodes sit at level I. They can either be linked to a load balancing switch with a single logical IP address, or they can be attached to hot-standby nodes for fault tolerance, with requests distributed by DNS. Static requests are processed locally at masters. Dynamic content requests may be processed locally at masters, or redirected to a slave node or another master. Slave nodes are at level II and are only used to process dynamic content requests. They may be non-dedicated and recruited dynamically when they become idle. If a slave node fails, a master node may need to restart a dynamic content process on another node. M/S separates dynamic and static content processing, so long running, resource intensive CGI scripts will not slow

down static content processing. Static requests require little processing time, but running them on the same node with dynamic requests may significantly increase their response time. Another disadvantage of mixing CGI with static content processing is that resource-intensive CGI requests tend to use a large amount of memory, which decreases space available for file system caching, further decreasing static request performance.

The M/S architecture has advantages over the flat architecture for fault tolerance and dynamic resource recruitment. In this paper, we are mainly interested in building an optimized M/S cluster which delivers better performance than a flat architecture without considering fault tolerance and dynamic resource recruitment. The experiment results in Section 5 show that without proper resource management, M/S can perform worse than an optimized flat architecture for a fixed number of nodes. Therefore, the main optimization issues that are considered in this paper are:

- Given p dedicated nodes in a cluster, how many nodes should be assigned as masters?
- Given a request, how should the system assign this request to one of master or slave nodes?
- Can an M/S architecture outperform a flat architecture (without considering fault tolerance and idle resource recruitment)?

In evaluating the proposed techniques, we use the stretch factor [21], which is the ratio of average response time with resource-sharing among requests to that without sharing, as the major performance comparison metric. More specifically, given a sequence of requests with execution times (or called service demands) as $d_1, d_2, \cdots, d_n$ and their request response times at the server site (the interval between request arrival and the end of processing) as $t_1, t_2, \cdots, t_n$, the stretch factor is

$$\frac{\sum_{i=1}^{n} t_i/d_i}{n}.$$

The goal of scheduling optimization is to minimize the stretch factor when n is very large. Internet delay between servers and clients is not included in response times because we are mainly interested in processing client requests as soon as possible at server sites. Also notice that there are other performance measurement metrics and evaluation results may be different with different criteria. Stretch factor is more suited for describing system performance because it relates a customer's waiting time to service demand [21]. It represents a trade-off that allows short Web requests not to be slowed down too much by large jobs. This choice can meet user's psychological expectation: in a system with highly variable task sizes, users may be willing to wait longer for large tasks to complete, but expect that small tasks should complete quickly [10, 6]. Furthermore, stretch factor reveals the load of a system. A system with a high stretch factor is obviously overloaded, but one with high response time may not be so because the high value may be due to actual long task service demand.

3 Analytic Modeling for Scheduling Optimization

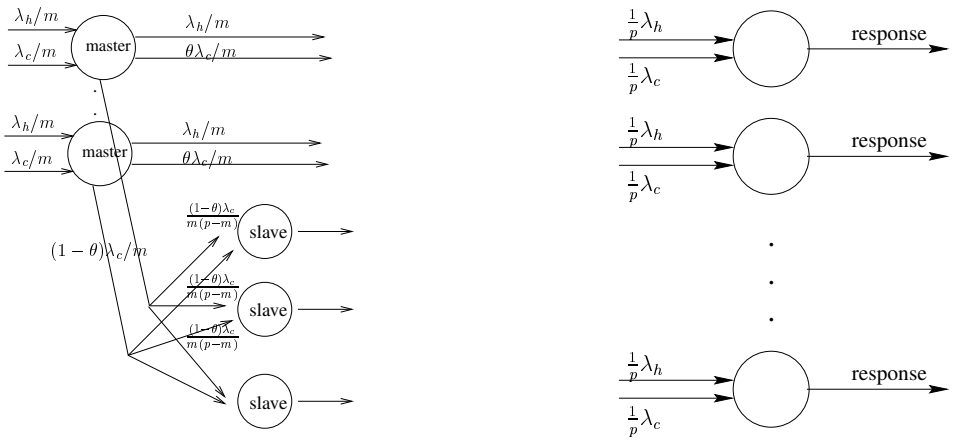
The main difficulty in designing a scheduling heuristic with near optimal performance is that the arrival interval between consecutive requests, the number of requests, and the request execution times are non-deterministic. Also, the service demand of each request involves mixed I/O and CPU activities and it is difficult to model such factors accurately. Our strategy is that we first make a number of assumptions in terms of request arrival intervals and execution time distribution, and conduct analytic modeling using queuing theory. Although the model we use here is relatively simple and certain practical aspects are ignored, our objective is to gain insights that can be used in the next section to guide the design of our scheduling policy and determine the number of master nodes that should be allocated.

We model the M/S and flat architectures as multiple class, open queuing networks. The two classes of customers to these two queuing systems are static requests and dynamic content requests. The servers in the two queuing networks are assumed to be homogeneous. Figure 2 depicts the two queuing network models. p is the number of dedicated servers in each cluster. m is the number of master nodes in the M/S architecture. And θ is the percentage of dynamic content requests processed locally at master nodes. Our goal is to find a range of θ and m such that the M/S architecture outperforms the flat architecture and determine the best value of θ and m that minimizes the stretch factor of M/S.

For the flat architecture, requests are randomly dispatched to nodes in the cluster with a uniform distribution. For the M/S system, requests are first randomly distributed among master nodes. The master nodes will process all static requests and redirect a portion of the dynamic content requests to slave nodes. We assume that the overhead for executing the remote dynamic content processing is negligible, because our implementation for such execution is very efficient [35].

The following terms describe the workload:

- λ_h, λ_c are the mean arrival rates of the two classes of customers, namely, static and dynamic content requests. For the flat architecture, requests are assumed to be evenly routed to nodes and the mean arrival rates of static requests and dynamic content requests to each server are λ_h/p and λ_c/p respectively. We use $\lambda = \lambda_h + \lambda_c$ to represent the total request arrival rate.
- μ_h and μ_c are the average service rates of static and dynamic content requests on each server. Note that service demand of a request is defined as the processing time of a request without resource contention from other requests. Service rate is defined as the reciprocal of service demand.
- Let $r = \frac{\mu_c}{\mu_h}$, and $a = \frac{\lambda_c}{\lambda_h}$. For most Web servers with extensive dynamic content generation, it is expected that $r << 1$.



(a) The M/S model. (b) The flat model.

Figure 2: Two clustering models.

We assume that the request queue length seen on average by an arriving customer must be equal to the time averaged queue length [25]. This assumption is satisfied by requests following Poisson distributions for arrival rates and for service times with exponential distribution. Requests can be processed in the First Come First Serve (FCFS) manner or processor sharing manner.

We further define the following terms:

- Let S_F be the stretch factor in the flat model. $S_{F,c}$ is the stretch factor for dynamic content requests only. $S_{F,h}$ is the stretch factor for static requests only.
- Let S_M be the stretch factor in the M/S model. $S_{M,c1}$ is the average stretch factor for dynamic content requests processed at the master nodes. $S_{M,c2}$ is the average stretch factor for dynamic content requests processed at slave nodes. $S_{M,h}$ is the average stretch factor for static requests only.

First, we are interested in choosing appropriate θ's and m's such that $S_M \leq S_F$. By applying the queuing network analysis technique, the following equations can be established for the M/S and flat architectures when the number of requests becomes extremely large (infinity):

$$\begin{array}{rcl} \frac{1}{\mu_h}/(1-\frac{\lambda_h}{m\mu_h}-\frac{a\theta\lambda_h}{mr\mu_h}) & = & S_{M,h}\times\frac{1}{\mu_h}, \\ \frac{1}{r\mu_h}/(1-\frac{\lambda_h}{m\mu_h}-\frac{a\theta\lambda_h}{mr\mu_h}) & = & S_{M,c1}\times\frac{1}{r\mu_h}, \\ \frac{1}{r\mu_h}/(1-\frac{a(1-\theta)\lambda_h}{r(p-m)\mu_h}) & = & S_{M,c2}\times\frac{1}{r\mu_h}, \\ \frac{1}{\mu_h}/(1-\frac{\lambda_h}{p\mu_h}-\frac{a\lambda_h}{rp\mu_h}) & = & S_{F,h}\times\frac{1}{\mu_h}, \\ \frac{1}{\mu_c}/(1-\frac{\lambda_h}{p\mu_h}-\frac{a\lambda_h}{rp\mu_h}) & = & S_{F,c}\times\frac{1}{\mu_c}. \end{array} \quad (1)$$

S_M and S_F can be computed as:

$$\begin{array}{rcl} S_M & = & \frac{\lambda_h S_{M,h}+\lambda_c\theta S_{M,c1}+\lambda_c(1-\theta)S_{M,c2}}{\lambda_h+\lambda_c} \\ & = & \frac{(1+a\theta)S_{M,h}+a(1-\theta)S_{M,c2}}{1+a}, \\ S_F & = & \frac{\lambda_h S_{F,h}+\lambda_c S_{F,c}}{\lambda_h+\lambda_c} = S_{F,h} = S_{F,c}. \end{array} \quad (2)$$

Using Equation 2, inequality $S_M \leq S_F$ can be transformed as:

$$(1+a\theta)S_{M,h} + a(1-\theta)S_{M,c2} \leq (1+a)S_{F,h}. \quad (3)$$

Using Equation 1, the above inequality can be furthered reduced to

$$\frac{1+a\theta}{1-\frac{\lambda_h}{m\mu_h}-\frac{a\theta\lambda_h}{mr\mu_h}} + \frac{a(1-\theta)}{1-\frac{a(1-\theta)\lambda_h}{r(p-m)\mu_h}} \leq \frac{1+a}{1-\frac{\lambda_h}{p\mu_h}-\frac{a\lambda_h}{rp\mu_h}}. \quad (4)$$

By eliminating the denominators in both sides of Equation 4, we have: $A\theta^2 + B\theta + C \leq 0$ where

$$\begin{array}{rcl} A & = & a^2\rho p(rp - r\rho + \rho) \\ B & = & a\rho(arm\rho - rm\rho - 2armp - am\rho + r^2p^2 - r^2pm \\ & & -pr^2\rho + r^2m\rho + rp\rho + rp^2 - rmp - ap\rho + apr\rho) \\ C & = & r^2m^2\rho - armp\rho + arm^2\rho + a^2m\rho^2 - ar^2pm\rho \\ & & +ar^2m^2\rho + a^2rm^2\rho + ar^2m\rho^2 - ar^2m\rho^2 \\ & & -a^2rm\rho^2 + r^2p^2\rho - 2r^2mp\rho - arp\rho^2. \end{array}$$

Thus the range of θ values that satisfy Inequality 3 can be obtained by solving the above inequality. Let θ_1 and θ_2 be the roots of equation $A\theta^2 + B\theta + C = 0$. Since $A > 0$, θ should be chosen in between θ_1 and θ_2.

Since coefficients A, B, and C are complicated, it is very difficult to derive θ_1 and θ_2 directly using the standard quadratic equation solving technique. We however notice that if $S_F = S_{M,h} = S_{M,c2}$, two sides of Inequality 3 become equal. The θ value for this case is $\frac{m}{p} - \frac{r(p-m)}{ap}$. Then we can let $\theta_2 = \frac{m}{p} - \frac{r(p-m)}{ap}$. Since $\theta_1 + \theta_2 = -\frac{B}{A}$, we have:

$$\theta_1 = -\frac{B}{A} - \theta_2 = \frac{m(a+1) - p + a\rho/r - a\rho}{ap - a\rho + a\rho/r}.$$

With the assumption that $m \geq \frac{rp}{a+r}$ and $r < 1$, it can be shown that $\theta_1, \theta_2 \leq 1$ and $0 \leq \theta_2$.

We can also prove that $\theta_1 \leq \theta_2$ by calculating the difference of θ_2 and θ_1. The difference $\theta_2 - \theta_1$ is

$$\frac{(p-m)(1-r)(p-\rho-a\rho/r)}{a(p-\rho+\rho/r)}$$

which is larger than or equal to 0 since $m \leq p, r < 1, \rho/r - \rho > 0$ and $1 - \rho/p - a\rho/(rp) > 0$. The last inequality holds because $S_{F,h} = 1/(1-\rho/p-a\rho/(rp)) > 0$ (c.f. Equation 1).

To choose the optimal m, we first choose the optimal θ's for each m and then choose the (m, θ) pair that gives the smallest average stretch factor for S_M. The best θ for each m is $\theta' = (\theta_1+\theta_2)/2$ which is the middle of the two roots. This value however may be less than 0 which is meaningless. Therefore we choose $\theta_m = \max(\theta', 0)$.

We cannot derive the closed form for calculating the optimal m. Instead, we calculate the stretch factor S_M numerically for each possible m value, then pick up the m which delivers the smallest stretch factor. Notice that the possible choices of m are integers between 1 and p.

The above results are summarized in the following theorem and they can also be extended for a heterogeneous system with non-uniform nodes.

Theorem 1 *Let* $\rho = \frac{\lambda_h}{\mu_h}$, $\theta_1 = \frac{m(a+1)+a\rho/r-a\rho-p}{ap-a\rho+a\rho/r}$ *and* $\theta_2 = \frac{m}{p} - \frac{r(p-m)}{ap}$. *If* $m \geq \frac{rp}{a+r}$ *and* $r < 1$, *then* $\theta_1 \leq \theta_2 \leq 1$, $0 \leq \theta_2$ *and* $\forall\theta \in [\theta_1, \theta_2]$, $S_M \leq S_F$. *The minimum value of* S_M *is achieved when* $\theta_m = \max((\theta_1 + \theta_2)/2, 0)$ *and the best* m *can be derived by numerically computing the following minimization problem:*

$$\min\{\frac{1+a\theta_m}{1-\frac{\rho}{m}-\frac{a\rho}{rm}} + \frac{a(1-\theta_m)}{1-\frac{a(1-\theta_m)\rho}{(p-m)r}}\}, \quad 1 \leq m \leq p.$$

This theorem gives a condition when M/S outperforms the flat architecture. It also gives a means for determining the best number of master nodes if the average request arrival and processing rates can be sampled in advance. We will apply these results in our system design and will evaluate its effectiveness in Section 5.

An alternative to the above M/S scheme is to fix the assignment of dynamic content requests to a few nodes but distribute static-content requests to all nodes. We call this scheme as M/S'. Similarly, we can derive the stretch factor for this scheme and we can show that this approach can also outperform the flat model, but is not as competitive as our M/S model. In Figure 3, we illustrate the improvement ratio of our M/S model to the flat model and to the M/S' model using the derived formula when the arrival rate $\lambda = 1000$, the number of processors p=32, the arrival rate ratio $a = 2/8, 3/7$ and $4/6$, and the processing rate ratio $r = 1/10, 1/20, 1/40$ and $1/80$. The reported data are $(\frac{Stretch-factor(Flat)}{Stretch-factor(M/S)} - 1) \times 100\%$ and $(\frac{Stretch-factor(M/S')}{Stretch-factor(M/S)} - 1) \times 100\%$. The results show that M/S outperforms the flat model by up to 60% and that it outperforms the M/S' model by up to 18%.

4 Scheduling Design with Practical Considerations

In a real environment, the availability of CPU and I/O resource changes dynamically, and also arrival rates and service rates of requests vary from time and time. For static Web requests that normally access small files, we do not re-schedule such requests since it only takes a very small amount of time to process. They will only be executed at master nodes. For resource-intensive dynamic content requests, our scheduler uses a prediction based model to determine which node is the best to run a dynamic content request, subject to resource reservations at master nodes to accommodate future static content requests.

In selecting the best node for dynamic content processing, we use periodically-updated I/O and CPU load information to estimate the expected cost for processing a dynamic content request and select a node with the minimum cost. This is important since I/O and CPU demand for different request types can vary significantly. We also reserve a certain amount of CPU and I/O for static content processing on each master node. In this way, master nodes are not overloaded with dynamic content processing and simple, static content requests can be processed promptly. Our theoretical analysis in Section 3 provides a guideline in controlling the percentage of dynamic content requests that should be processed at master nodes.

Node selection with cost prediction. Without knowing the CPU and I/O demand of each request, it would be hard to use I/O and CPU load information. Also, for some dynamic content requests such as database searching it is hard to provide an accurate formula to predict the cost. We use the following formula to estimate the *relative server-site response cost* (RSRC) of each dynamic content request on a node in a homogeneous cluster:

$$RSRC = \frac{\omega}{CPUIdleRatio} + \frac{(1-\omega)}{DiskAvailRatio}. \qquad (5)$$

The meanings of these terms in Equation 5 are described as follows. ω is the average percentage of the cost contributed by the CPU for a request. $1-\omega$ is used to approximate the average percentage of the cost contributed by I/O. ω is obtained by off-line sampling, approximating the I/O and CPU demand of the request on an unloaded system at a Web site. If a value for ω cannot be obtained, we assume $\omega = 0.5$, which means that I/O and CPU resources are considered to be equally important. $CPUIdleRatio$ is the percentage of idle CPU time available for usage in this node. $DiskAvailRatio$ is the available ratio of disk bandwidth in this node. These two numbers change dynamically based on the system load and are supplied at runtime to compute the RSRC. In our implementation, we use the Unix rstat() function to collect the load information on each node.

It should be noted that disk caching is not considered in our model because of the modeling difficulties. However, our prediction provides a reasonable indication of resource demands. If nodes in a cluster are non-uniform, the relative speed in accessing CPU

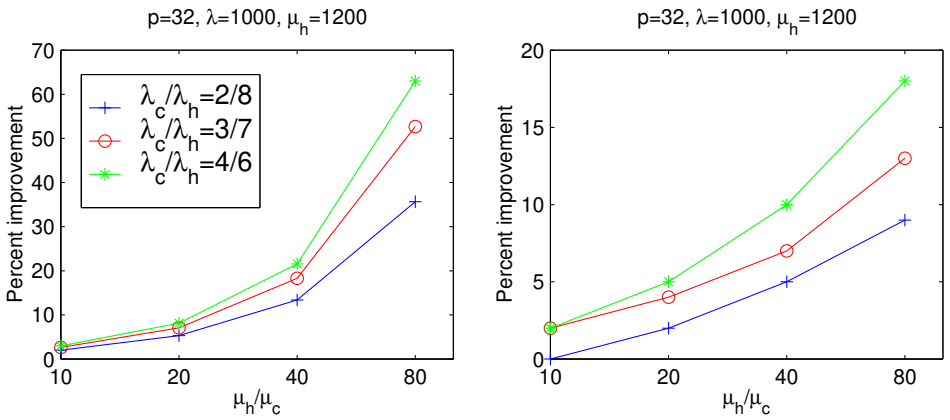


Figure 3: (a) Improvement of M/S over the flat model. Notice that $\frac{\mu_h}{\mu_c} = \frac{1}{r}$. (b) Improvement of M/S over the M/S' model.

and disk I/O resource needs to be considered. This issue has been addressed in our previous work [36] and will not be discussed here.

Reservation for static request processing. Ideally, the percentage of dynamic content requests processed at masters should be θ_m from Theorem 1 of Section 3. In order to be more effective, this term requires an accurate estimation of service rates of dynamic content and static requests during a short period of time, which is difficult. In our scheme, we set a limit on the percentage of dynamic content requests that should be processed at masters. Since the upper bound for M/S to outperform the flat architecture is θ_2 as defined in Theorem 1 of Section 3, we set the reservation ratio at $\theta'_m = \frac{m}{p} - \frac{r(p-m)}{ap}$. During execution, the percentage of dynamic content requests scheduled to masters may not reach this limit. Note that this limit only depends on the *relative* service rates of dynamic content and static requests (r), and the relative arrival rates of dynamic content and static requests (a). We monitor the arrival rates for calculating a. However, it is not easy to provide an accurate online estimation of request service rates. As a compromise, we use current *relative response times* of static and dynamic content requests to approximate r. The load managers on master nodes update θ'_m periodically by collecting response times of requests on different servers.

Note that the adjustment of θ'_m is self-stabilizing in the sense that θ'_m will converge to a specific value if the system itself is stable, no matter what the initial value of θ'_m is. For example, if the initial value of θ'_m is too low, very few dynamic content requests will be processed by masters. Then r will decrease, θ'_m will increase and more dynamic content requests will be executed on masters. When θ'_m becomes too large, too many dynamic content requests are executed at master nodes, which slows down processing of static requests, thus r will increase which will lower θ'_m. The value of θ'_m is also adjusted according to ratio a. With more static requests compared to dynamic content requests, the ratio a and θ'_m will also decrease. Thus, more resources are reserved for static processing at master nodes.

5 Experiments

The prototype implementation of our method is based on the freely available Apache Web server version 1.3 source code and the Swala server with cooperative content caching support [17]. In this section we discuss the workload in our trace-driven experiments, the method to replay the traces, and then present experimental results. The protocol for dynamic content generation is CGI.

5.1 Workload description and evaluation methodology

Table 1 summarizes the main characteristics of the traces we have obtained. The UCB log [16] was gathered from the Home IP service offered by UC Berkeley to its modem pool users. The KSU log was recorded by the Web server for online library service at Kansas State University. The ADL log is from the testbed of Alexandria Digital Library [3], a digital library for spatially referenced data. For these logs we can distinguish dynamic and static requests, and can extract the completion time of each request and the response size. We also have a DEC trace [24] available. For both UCB and DEC traces we cannot recognize URLs and their parameters because they are scrambled for privacy concern. Since the dynamic content request percentage of the DEC trace is similar to the UCB log, we decided not to use that trace. To keep trace replay time reasonable, we extracted a segment of the UCB log for use in our tests. The extracted segment consists of 128668 requests and has a time span of 4 hours.

Log name	Year	No. requests	% CGI	Average Interval	HTML size	CGI size
DEC	1996	24.5 M	8.7	0.09 s	8821	5735
UCB	1996	9.2 M	11.2	0.139	7519	4591
KSU	1998	47364	29.1	18.486	482	8730
ADL	1997	73610	44.3	22.418	2186	2027

Table 1: Characteristics of four Web traces.

Our tests are performed by resubmitting all the requests in each log to our server cluster. To replay requests, two major issues are considered:

- **Request arrival intervals.** If we issue requests to our server cluster with the same rates as recorded in the logs, the load will be too light for a tested cluster. To show the benefits of clustering in processing heavy loads, we scale intervals among requests so that requests in each log are issued to the cluster at various fast rates to represent different degrees of loads. In this way, we can study the behavior of popular, CGI-intensive Web sites. [1]

- **Replaying an individual request**. If a request is to retrieve a file, replaying it is relatively simple since the recorded result size is the file size. Since our research is targeted at Web sites with intensive CGI processing, variation in static file access patterns will not dramatically affect overall system behavior. We replace all file fetches from the logs with the 40 representative files from SPECWeb96. For each file request in the log, the file in this set with the closest size is returned.

  For CGI requests, replaying is more difficult. For the KSU log, we do not have its proprietary library data and binary, and we cannot determine the request service time from the log data. For the UCB log, we do not know what each scrambled CGI request is doing, and we cannot determine the request service time either. We will need to replace those CGI requests with real operations in order to run experiments. In doing such a replacement, we generate synthetic loads which represent CPU and I/O intensive or mixed situations. For the UCB trace, we use a CGI script from the WebSTONE benchmark [33]. This script receives a file size as a parameter and dynamically generates a file of that size and returns it to the client. We modified this script so that we can control the running time of the script in order to generate different loads by CPU busy-spinning. As a result, these CGI requests are CPU intensive. For the KSU library-searching requests, we created a database with approximately 10000 items using the WebGlimpse software [27] and replaced the CGI library requests with WebGlimpse commands. These requests have mixed CPU and I/O demands, but on average 90% of service time is spent searching index information in memory. For the ADL trace, we replicated a small ADL catalog database in our local cluster and ran an ADL server at each node. This workload is I/O intensive with about 90% of the servicing time consumed by disk accesses.

To study system performance on a large cluster for processing different workloads, we have developed a simulator of a Web server cluster which approximates the behavior of OS management for CPU, memory and disk storage. This simulator consists of a UNIX-like job scheduler, an access scheduler for storage and a virtual memory manager. Each Web request will be provided with service time, I/O and CPU demand distribution. Each request job will be modeled as a sequence of CPU bursts and I/O bursts, submitted to the CPU queue and I/O queue. CPU scheduling is based on the UNIX BSD 4.3 strategy [26]. The process ready queue is a multilevel feedback queue divided into multiple lists according to process priority. Processes are scheduled based on priority and may be preempted following quantum expiration. The I/O queue also maintains a set of I/O processes and is scheduled using round-robin. The memory management maintains a set of free pages and allocates a number of pages to a new process. For each request, a memory size requirement is provided and the system generates working-set oriented access patterns to stress the demand-based paging scheme. The above setting represents fairly simple OS functions in managing CPU, memory and I/O. Network contention within a cluster is not modeled because it rarely happens for dynamic content-intensive processing [35]. Other factors such as disk caching are also not considered. We believe that such a simulator is representative of the behavior of modern OS systems in processing Web workloads. In Section 5.2.2, we compare simulated results and experimental results. The difference is reasonably small.

5.2 Experimental results

In our experiments, we assess the effectiveness of the proposed optimization for M/S and isolate improvement ratios due to the use of cost sampling, master reservation, and separation of static and dynamic content processing. Since the number of master nodes in M/S is based on sampled arrival and service ratios of static and dynamic content requests, we also examine the performance sensitivity of our setting when these rates vary dramatically during execution. Finally to ensure the quality of simulation, we have also conducted a validation on a SUN cluster to compare simulated and actual results.

5.2.1 Simulation results

Parameter setting. The numbers of nodes in the simulated clusters are 32 and 128 which represent medium and large cluster sizes for current and future Web workloads. Each node has the capability to handle 1200 requests/second for SPECWeb96 based on the results submitted to SPEC from 1996 to 1998. Those rates may be higher than what average workstation servers can deliver; however, these numbers are reasonable for a commercial Web site or for future machines, considering advances in CPU, disk, memory and networking technology. The system overhead charged in the simulation is based on current high-end server performance [28]. The CPU quantum is 10milliseconds. The priority update period is 100 milliseconds. The context switch overhead is 50 microseconds. The fork overhead is 3 milliseconds. The remote CGI latency (excluding fork) is 1 millisecond, representing TCP connection time. The page size is 8KB. The I/O burst for accessing a page is an average of 2 milliseconds. Notice that disk seek time for a sector is longer than 2 milliseconds; however, considering that current stor-

[1] Currently Yahoo receives 115 million page views per day in June 1998 [34] which is around 1331 hits per second. We expect the hit ratios will increase substantially over next few years.

age speed can range from 5 to 80MB/second, with support of file caching and block transfer, it is reasonable to use this number.

Table 2 lists the workload parameters examined for the three traces. The average ratio of the CGI arrival rates to static request rates (a) is fixed, based on the log data. Arrival rates (λ) are scaled in replaying to reflect various workloads. Arrival rates considered vary from 500 to 8000 requests/second, to reflect a current or future popular Web site. Since KSU and ADL traces have much more CGI activity than UCB, and since 128 nodes can handle four time more load than 32 nodes, the arrival rates we have examined for each trace are listed in Table 2 and such a setting creates reasonable loads for a corresponding cluster to handle. Otherwise, the load would be too light or too heavy. The average ratio of CGI processing rate to static request rate, r, is chosen to be: $1/20, 1/40, 1/80, 1/160$ to represent a wide range of CGI resource demands. This choice is based on the previous studies [20, 17, 36].

	a	Arrival rates	r
UCB	0.12	1K, 2K for 32 nodes. 4K, 8K for 128 nodes.	$\frac{1}{20}, \frac{1}{40}, \frac{1}{80}, \frac{1}{160}$
KSU	0.42	0.5K,1K for 32 nodes. 2K, 4K for 128 nodes.	$\frac{1}{20}, \frac{1}{40}, \frac{1}{80}, \frac{1}{160}$
ADL	0.78	0.5K,1K for 32 nodes. 2K, 4K for 128 nodes.	$\frac{1}{20}, \frac{1}{40}, \frac{1}{80}, \frac{1}{160}$

Table 2: Workload parameters examined.

Effectiveness of the proposed optimization. We call our optimization scheme M/S and compare it with the following alternative solutions in order to examine the effectiveness of each optimization strategy used in M/S: 1) **M/S-ns.** No sampling is used to assess I/O and CPU demands. We set $\omega = 0.5$, assuming that I/O and CPU are of equal importance for CGI requests. Performance gain reflects the benefits of demand sampling. 2) **M/S-nr.** No reservation is used to keep a portion of master resources available for static content processing. Performance gain reflects benefits of resource reservation in avoiding overloading of master nodes. 3) **M/S-1.** This setting treats all nodes as master nodes and the scheduling algorithm however is not changed. Performance gain reflects the benefits of separating static and CGI requests. We report the percentage of improvement: $(\frac{Stretch_factor(M/S-?)}{Stretch_factor(M/S)} - 1) \times 100\%$ where "?" stands for "ns", "nr" or "1". Figure 4 shows the results of the comparison.

Figure 4 shows that M/S has significant improvements over M/S-nr (up to 68% reduction), which underscores the importance of reserving resources for static requests. The figure also shows that M/S-1 could have up to 26% performance degradation, which reveals the benefits of separating static requests from CGI requests. The performance improvement of M/S over M/S-ns ranges from 5% to 22% and the average is 14%, which shows that sampling characteristics of CGI requests is useful. M/S-1 can actually be viewed as a flat architecture with remote CGI. If we compare M/S-1 and M/S-nr, it is clear that without resource reservation on master nodes, M/S can perform worse than a properly configured flat architecture. This means that without considering fault tolerance and idle resource recruitment, M/S may not be competitive to a flat architecture if optimization is not properly applied.

Sensitivity analysis on number of masters. Our method for determining the number of master nodes is based on sampled average service rates of CGI and static content requests and their arrival rates. Such a number may be stable for a certain period, e.g. on a weekly basis and the number of master nodes can be changed by administrators periodically. If fluctuations within request patterns are very high, parameter estimation may be poor. Ideally M/S should be able to choose the numbers of master nodes based on dynamically changing workload characteristics. In this experiment, we study the system performance sensitivity and fix the number of master nodes m decided by using parameter $r = 1/60$, $a = 0.44$, $\lambda = 750$ for 32 nodes and $\lambda = 3000$ for 128 nodes. The calculated number of masters is 6 for 32 nodes and 25 for 128 nodes. Then we use this configuration to handle the three traces in which r varies from 1/20 to 1/160, ratio a varies from 0.12 to 0.78, λ varies from 500 to 2000 for for 32 nodes, and from 2000 to 8000 for 128 nodes. Figure 5 shows increasing ratios of stretch factors of M/S using the fixed m compared to the one which adapts to runtime parameters change. We can see that the degradation ratio is at most 9% and the average is 4%. This means that performance of M/S with fixed m is fairly robust. M/S with a variable number of masters may be better; however, it requires dynamic configuration change.

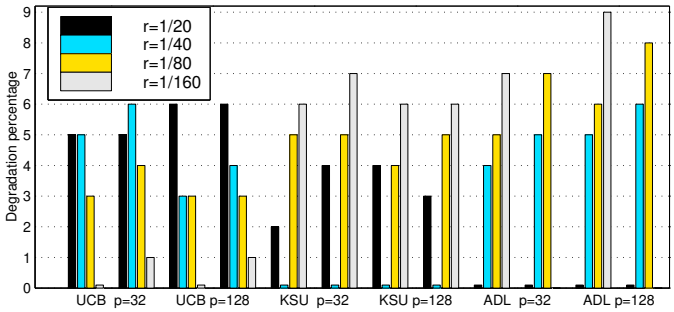


Figure 5: Performance degradation when using a fixed number of masters. The arrival rates for the 12 bar groups are 1000/s, 2000/s, 4000/s, 8000/s, 500/s, 1000/s, 2000/s, 4000/s, 5000/s, 1000/s, 2000/s and 4000/s

5.2.2 Validation on a Sun cluster

We have also implemented and run our M/S Web server prototype on a Sun cluster (Solaris 2.5) containing six Sun Ultra I workstations connected by a Fast Ethernet switch to validate conclusions drawn from simulated results. With SPECWeb96 benchmark, each Ultra 1 node can process static requests at 110 requests/second.

The ADL trace is replayed with a database smaller than what is used at its real site due to storage constraints. The ratio r is about 1/40. For the UCB trace, we ran each CGI using a CPU-intensive script with an average ratio $r = 1/40$. We ran each KSU CGI using a WebGlimpse search script with $r = 1/40$. The arrival rates used are 20 requests/second and 40 requests/second. Requests are sent

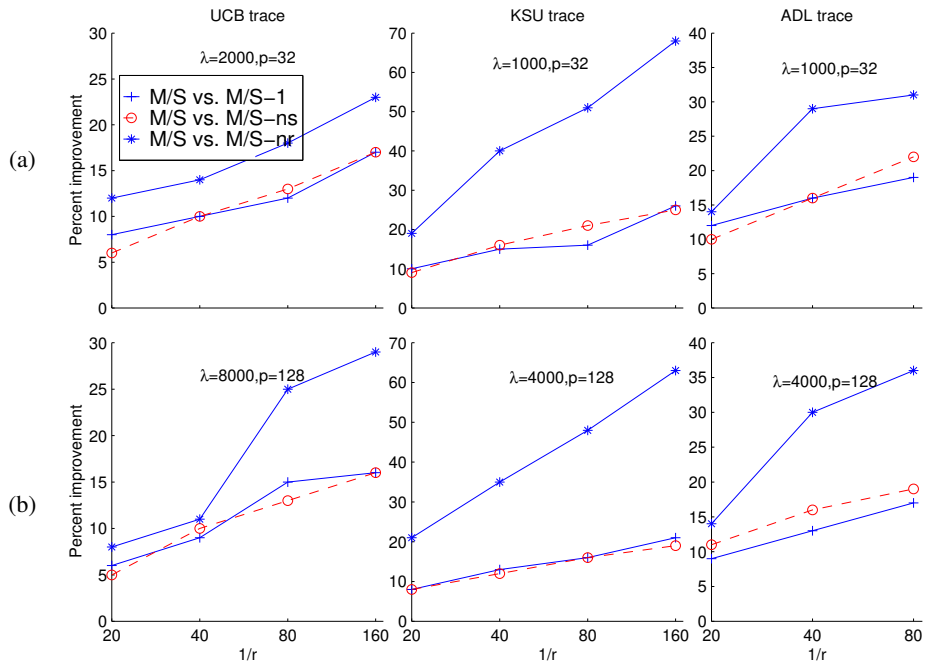


Figure 4: Percentage of improvement using different optimization strategies in M/S. (a) p=32. (b) p=128.

to servers in a round-robin fashion. The number of master nodes are 3, 1 and 1 for the UCB, KSU and ADL traces, respectively.

Simulated vs. actual improvement ratios. Table 3 shows the improvement ratio of the M/S method over other methods derived by simulation and by actual execution. Again, the simulated performance and experimental results are very close for all cases and the average difference is around 3%. Simulated results are slightly optimistic, compared to actual performance. This is because the simulator does not consider background jobs running in the cluster and only captures approximated behavior of Solaris OS 2.5.

Trace & λ	M/S vs. M/S-1		M/S vs. M/S-ns		M/S vs. M/S-nr	
	Actual	Simu	Actual	Simu	Actual	Simu
UCB, $20/s$	07%	05%	08%	09%	14%	12%
UCB, $40/s$	12%	10%	13%	14%	18%	19%
KSU, $20/s$	10%	12%	09%	12%	23%	27%
KSU, $40/s$	17%	20%	19%	18%	32%	35%
ADL, $20/s$	12%	13%	13%	15%	24%	27%
ADL, $40/s$	17%	19%	16%	19%	37%	40%

Table 3: Performance improvement of M/S over other methods on a SUN cluster by actual running and simulation.

6 Concluding Remarks and Related Work

In summary, we have studied a master/slave scheduling framework integrated with necessary optimization strategies for clustering Web servers. Within a two-level framework, proper optimization in resource management is needed and our experiments show that the proposed methods can achieve up to a 68% performance improvement compared to the alternatives. The theoretical results that compare an M/S model and a flat model have made a number of simplifying assumptions, but they do provide useful insight on the behavior of two models and guide our scheduler design. The experimental results demonstrate the effectiveness of such a design. The presented results are focused on a homogeneous cluster and we are making an extension for managing heterogeneous nodes.

Our design and experiments assume that data of a Web server is either fully replicated or stored on a shared device and CGI scripts can run on any server. In practice, only portions of the data may be replicated and some CGI scripts require specific servers to run. Scheduling design becomes more challenging and application-specific. We plan to study this issue in the future.

Inktomi [18] and AltaVista [12] have used an M/S architecture for organizing their searching engines. A generalization from the Inktomi server to a layered structure is studied in [15] for network services including proxy distillation. Their work is focused on system modeling in organizing a cluster for general network services and there is no detailed study and performance evaluation on issues specific to Web servers. Our contribution is to provide theoretical analysis and heuristic design to optimize request scheduling in an M/S architecture. In this way, Web server software developers and application users can benefit from our studies.

In addition to DNS based clustering solution for Web servers [5, 9, 19, 22], there is another project in [11] proposes a routing solution with single IP, similar to the switching techniques used in [8, 13, 14]. Notice that previous work on general distributed

load balancing [32] normally considers one load index. Our earlier SWEB work [5] uses multiple resource load indices and request redirection. The techniques proposed in this paper improves SWEB by using low-overhead remote dynamic content processing instead of request redirection. Also our new scheme employs simple cost approximation for request scheduling while SWEB requires exact cost prediction, which is not feasible for general Web requests such as database searching.

Web caching for static contents at a server cluster is considered in [30] and dynamic content processing is not their focus. Web caching for dynamic content is possible if content is not changed frequently and this issue is studied in our Swala Web server [17]. Our work in this paper does not consider CGI caching. We actually have implemented our testbed on Swala and a simple extension to consider caching in our scheme can be incorporated. Request scheduling for static content processing based on file sizes is studied in [10] where short jobs for accessing small file sizes are assigned to the lighted loaded nodes to avoid mixing with heavy jobs which access large files. While this work does not address dynamic contention generation, our work is reminiscent in the sense that short file access jobs are assigned to masters and long CGI jobs are mainly assigned to slaves.

Acknowledgments

This work was supported in part by NSF CCR-9702640, CDA-9529418, and IRI94-11330 (Alexandria Digital Library project). We would like to thank Anurag Acharya and anonymous referees for their helpful comments and suggestions, Dan Andresen and Tim McCune for providing us the KSU trace, and Vegard Holmedahl for implementing the Swala Web server.

References

[1] A. Acharya, G. Edjlali, and J. Saltz. The utility of exploiting idle workstations for parallel computation. *ACM SIGMETRICS'97*, pages 225–236, 1997.

[2] T. E. Anderson, D.E. Culler, D.A. Patterson, and the NOW Team. A case for NOW(networks of workstations). *IEEE Micro*, February 1995.

[3] D. Andresen, L. Carver, R. Dolin, C. Fischer, J. Frew, M. Goodchild, O. Ibarra, R. Kothuri, M. Larsgaard, B. Manjunath, D. Nebert, J. Simpson, T. Smith, T. Yang, and Q. Zheng. The WWW Prototype of the Alexandria Digital Library. In *Proceedings of ISDL'95: International Symposium on Digital Libraries*, August 1995. Revised version appeared in IEEE Computer, 1996, No.5.

[4] D. Andresen, T. Yang, O. Egecioglu, O. Ibarra, and T. Smith. Scalability Issues for High Performance Digital Libraries on the World Wide Web . In *Proc. of IEEE ADL '96 (Advances in Digital Libraries)*, pages 139–150, May 1996.

[5] D. Andresen, T. Yang, V. Holmedahl, and O. Ibarra. Sweb: Towards a scalable WWW server on multicomputers. *Proc. of Intl. Symp. on Parallel Processing, IEEE*, pages 850–856, April 1996.

[6] Michael Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and Stretch Metrics for Scheduling Continuous Job Streams. In *Proceedings of Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1998.

[7] H. Casanova and J. Dongarra. Netsolve: a network server for solving computational science problems. *Proceedings of Supercomputing'96*, November 1996.

[8] CISCO. *Local Director*. http://www.cisco.com/warp/public/751/lodir/index.shtml, 1997.

[9] Michele Colajanni, Philip S. Yu, and Daniel M. Dias. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 585–598, June 1998.

[10] M. E Crovella and M. Harchol-Balter. Task assignment in a distributed system: Improving performance by unbalancing load. *Proc. of ACM SIGMETRICS'98*, pages 268–269, July 1998.

[11] Om P. Damani, P.Emerald Chung, Yennun Huang, Chandra Kintala, and Yi-Min Wang. One-IP: Techniques for Hosting a Service on a Cluster of Machines. In *Proceedings of the Sixth Int. World Wide Web Conference*, April 1997.

[12] Digital Equipment Corporation. *About Alta Vista*. http://www.altavista.com/av/content/about.htm, 1995.

[13] F5Labs. *BigIP*. http://www.f5.com/, 1997.

[14] Foundry Networks. *ServerIron Server Load Balancing Switch*. http://www.foundrynet.com, 1998.

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

[16] S.D. Gribble. *UC Berkeley Home IP HTTP Traces*. http://www.acm.org/sigcomm/ITA/, 1997.

[17] V. Holmedahl, B. Smith, and T. Yang. Cooperative Caching of Dynamic Content on a Distributed Web Server . In *Proc. of Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 243–250, July 1998.

[18] Inktomi Corporation. *The Inktomi Technology Behind HotBot, a White Paper*. http://www.inktomi.com, 1996.

[19] A. Iyengar and J. Challenger. Improving Web Server Performance by Caching Dynamic Data . In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[20] A. Iyengar, E. MacNair, and T. Nguyen. An analysis of web server performance. *Proceedings of IEEE GLOBECOM*, pages 1943–1947, November 1997.

[21] Raj Jain. *The Art of Computer Systems Performance* . Academic Press, 1992.

[22] E.D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: the NCSA prototype. *Computer Networks and ISDN Systems*, 27:155–164, 1994.

[23] K.Dincer and G. C. Fox. Bulding a world-wide virtual machine based on Web and HPCC technologies . In *Proceedings of ACM/IEEE SuperComputing'96*, November 1996.

[24] T.M. Kroeger, J. Mogul, and C. Maltzahn. *Digital's Web Proxy Traces*. ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html, 1997.

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

[26] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *The Design and Impelmentation of the 4.3BSD UNIX Operating System*. Addison Wesley, 1990.

[27] U. Manber, M. Smith, and B. Gopal. Webglimpse – combining browsing and searching. *Proceedings of the Usenix Technical Conference*, January 1997.

[28] Larry McVoy and Carl Staelin. lmbench: Portable tools for performance analysis. *Proceedings of the Usenix Technical Conference*, January 1996.

[29] NCSA. *Common Gateway Interface*. http://booboo.ncsa.uiuc.edu/cgi/, 1995.

[30] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. Locality-Aware Request Distribution in Cluster-based Network Service. In *Proceedings of ASPLOS-VIII*, October 1998.

[31] G. F. Pfister. *In Search of Clusters*. Prentice Hall, 1998.

[32] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE CS Press, 1995.

[33] G. Trend and M. Sake. WebSTONE: The first generation in HTTP server benchmarking. *Silicon Graphics, Inc. whitepaper: http://www.sgi.com/*, February 1995.

[34] Yahoo! Inc. *Yahoo! Investor Relations Center*. http://www.yahoo.com/info/investor/, 1998.

[35] H. Zhu, B. Smith, and T. Yang. A Scheduling Framework for Web Server Clusters with Intensive Dynamic Content Processing. Technical Report TRCS-98-29, CS Dept, UCSB, November 1998. http://www.cs.ucsb.edu/research/rcgi.

[36] H. Zhu, T. Yang, Q. Zheng, D. Watson, O. H. Ibarra, and T. Smith. Adaptive load sharing for clustered digital library servers. *Procecedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 235–242, July 1998.