Comparison of Load Balancing Strategies on Cluster-based Web Servers

Yong Meng TEO
Department of Computer Science
National University of Singapore
3 Science Drive 2
Singapore 117543
email: teoym@comp.nus.edu.sg

Rassul AYANI
Department of Microelectronics and Information Technology
Royal Institute of Technology (KTH)
164 40 Kista, Stockholm
Sweden

Abstract

This paper focuses on an experimental analysis of the performance and scalability of cluster-based web servers. We carry out the comparative studies using *two* experimental platforms, namely, a *hardware testbed* consisting of 16 PCs, and a trace-driven discrete-event *simulator*. Dispatcher and web server service times used in the simulator are determined by carrying out a set of experiments on the testbed. The simulator is validated against stochastic queuing models and the testbed. Experiments on the testbed are limited by the hardware configuration, but our complementary approach allows us to carry out scalability study on the validated simulator. The three dispatcher-based scheduling algorithms analyzed are: *round robin* scheduling, *least connected* based scheduling, and *least loaded* based scheduling. The least loaded algorithm is used as the baseline (upper performance bound) in our analysis and the performance metrics include average waiting time, average response time, and average web server utilization. A synthetic trace generated by the workload generator called SURGE, and a public-domain France Football World Cup 1998 trace are used. We observe that the round robin algorithm performs much worse in comparison with the other two algorithms for low to medium workload. However, as the request arrival rate increases the performance of the three algorithms converge with the least connected algorithm approaching the baseline algorithm as at a much faster rate than the round robin. The least connected algorithm performs well for medium to high workload. At very low load the average waiting time is two to six times higher than the baseline algorithm but the absolute value between these two waiting times is very small.

1. Introduction

The exponential growth of the Internet and its applications in the recent years has created the need for faster web servers to reduce the response time and provide better service to the users. An alternative to a powerful mainframe would be a cluster of processors as web server. A survey of scalable web server clustering architectures is discussed in [21]. An important issue is the load balancing scheme adopted, which influenced the performance and scalability of such architectures. A load balancing scheme in a cluster-based web server can

be divided into two parts. This consists of the *entity* that performs the load-balancing task, and the *algorithm* used to make decision in distributing HTTP requests among the servers.

Based on the entities that perform the load balancing, Cardellini et al. classify load balancing schemes into four main approaches: *client-based*, *DNS-based*, *dispatcher-based* and *server-based* [6]. In this paper we focus on dispatcher-based clusters, where one of the processors (the dispatcher) receives all incoming requests and distributes them among the servers. Dispatcher-based web-server cluster systems include IBM Network Dispatcher [12], Cisco Local Director [8], and Linux Virtual Server Project [16].

The SWEB architecture proposes a round-robin DNS policy as a first-level load balancing, and a second-level asynchronous scheme based on redirection [1]. The main disadvantages of DNS-based schemes are that it controls only part of the incoming requests. It is reported that DNS caching introduces skewed load on a clustered server by an average of ±40 percent of the total load [5]. In addition, HTTP redirection increases user's response time, since each redirected request requires a new client-server connection. Round-robin DNS was found to be of limited value and the research described in [9, 19] quantifies these limitations. To alleviate the performance loss due to redirection of HTTP requests at the DNS, Bryhni et al. proposed redirection of request in the network [5]. The four load balancing algorithms used in their simulation study are round robin, least connection, round trip time based on an average window size of one second, and Xmitbyte that is similar to round trip but is based on the amount of bytes transmitted. However, the study is limited to a small cluster of 4 servers.

Many cluster-based web-server architectures employ a simple Round-Round (RR) job (client HTTP requests) scheduling algorithm, which is simple to implement, but it is often inefficient. A better approach would be to weight the server workload in the scheduling scheme. For instance, the dispatcher may assign the next request to the least loaded (LL) server or to the server with the least connections (LC). As it will be discussed in the next section, the LL algorithm (referred to as the baseline algorithm) provides the best performance but it requires detailed information about the workload and often cannot be used in practice.

The aim of this investigation is to study the *performance* and *scalability* of dispatcher-based web server clusters. We compare the performance of three scheduling algorithms (RR, LC, and LL) with respect to *response time* and *scalability*. For this purpose we have used a cluster of web servers consisting of 16 Pentium II PCs running Linux Redhat 6.2. We studied the load balancing issues on our testbed and measured the service times of the servers and the dispatcher. The testbed also provided us with the opportunity to understand the details of using a cluster of computers as a web server. However, we also recognized several limitations of the testbed, such as difficulties to change the system parameters. Hence, we developed a discrete-event simulator to study the scalability of the cluster-based web servers. Obviously, a simulator is more flexible than a hardware platform. It is easier to increase the number and computation power of the servers in a simulator than in an existing hardware platform.

As will be discussed in the paper, the test-bed was used to validate the simulator (by comparing their results) and to provide input data (such as dispatcher service time) for the simulator. Section 2 discusses the modeling of a dispatcher-based cluster of web server and three scheduling algorithms. In section 3, we explain how we model and measure the

dispatcher service time and *web server service time*. Section 4 discusses the two workload traces used in this paper. Section 5 presents the validation of our trace-driven simulator. Section 6 discusses the experiments, and performance and scalability analysis of the three algorithms. The observations and conclusions are summarized in section 7.

2. Load Balancing Schemes

A URL requests sent to a web server is processed in two major steps:

a. the URL is translated into an IP address by a domain name server (DNS), and
b. the client uses the IP address of the destination server to submit its HTTP request to the server.

In a cluster-based server, one of the main issues is the assignment of URL requests to different servers. This can be done at the client side, as the Netscape Navigator does it. Over 100 web servers support the Netscape's web site. Each time the browser wants to visit the Netscape site, it randomly selects one server from the server pool. This in effect is a random load distribution. Another approach would be to perform the load distribution at the DNS server. In this method, the DNS server selects one of the servers belonging to the cluster, assuming that all the servers are connected to the WAN and each server has its own IP address. Indeed, some early web server clusters use this scheme (see [6] for details and the related problems). A third approach that is adopted in our work is dispatcher-based work distribution.

2.1 Dispatcher-based Web-Server Cluster

As the most popular choice of multiple-server cluster, the dispatcher-based architecture is featured by a processor called *dispatcher* that is responsible for receiving all the incoming client requests and distributing them among the back-end servers. An overview of this architecture is shown in Figure 1.

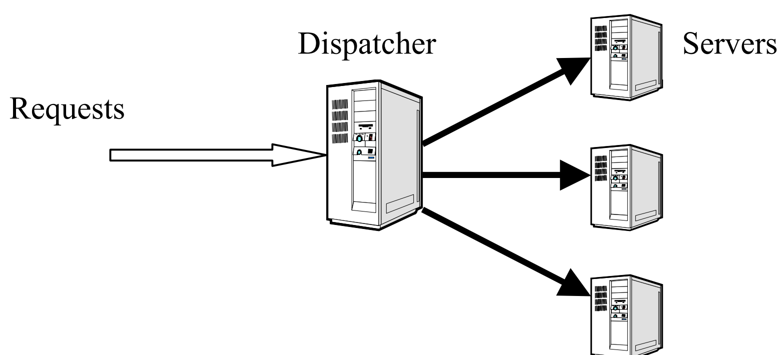

Figure 1: High-level View of a Dispatcher-based Server Cluster

In recent years, several academic prototypes and industrial products have employed the dispatcher-based approach, e.g., Magirouter (Berkeley 1996 discussed in [1]), LARD (Rice Uni. 1998, discussed in [20]), Linux Virtual Server (GNU project, 1998, discussed in [16]), eNetwork Dispatcher (IBM 1996, see [12]), and ONE-IP (Bell Lab 1996, discussed in [10]).

Implementations of dispatcher-based web cluster can be classified according to the network technique that the dispatcher uses to distribute the requests to the clients, as discussed in [16]. Actually, the dispatcher-based technique may be implemented in three different ways: *Network Address Translation (NAT), Direct Routing (DR)* and *IP Tunneling (IPTun)*[16]. In the NAT, both the data from the client to the cluster server (incoming data) and the data from the server to the client (outgoing data) will go through the dispatcher. But in DR and IPTun the incoming data pass the dispatcher, whereas the outgoing data is directly sent to the clients, as illustrated in Figure 2.

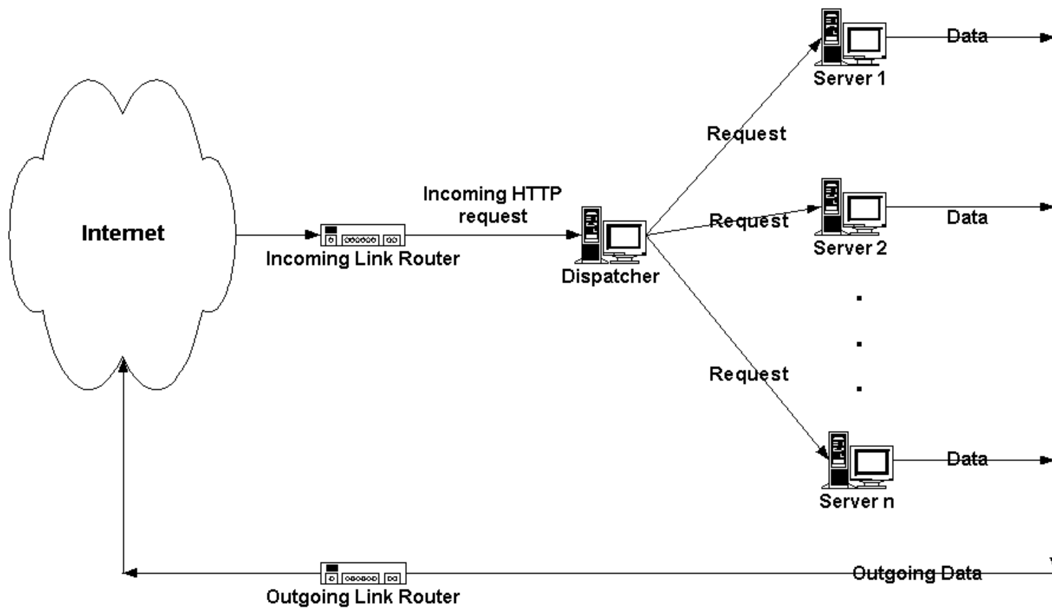

Figure 2: A Dispatcher-based Cluster of Web Server

In this paper, we assume that the outgoing data does not pass through the dispatcher and focus only on distribution of the incoming data to a number of homogenous servers. Thus, the cluster we are studying can be abstracted as a queuing system shown in Figure 3.

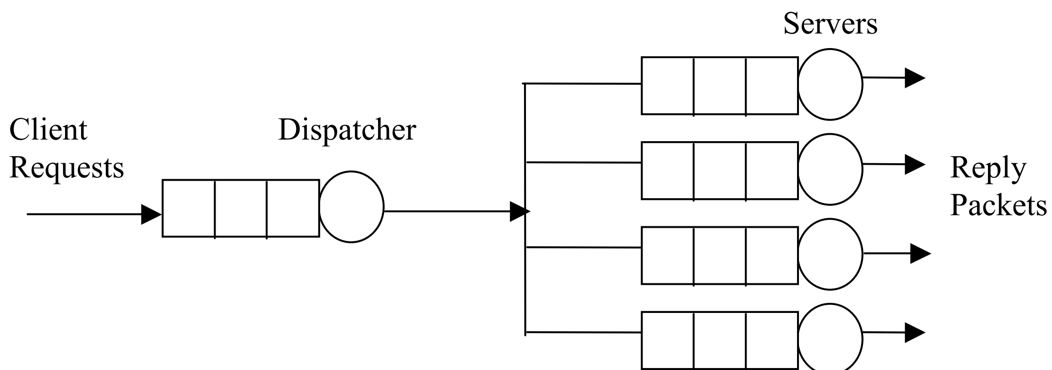

Figure 3: Queuing Model of a Dispatcher-based Cluster of Web Servers

As figure 3 suggests, we assume data replication (e.g., a copy of all files are accessible by each server). Moreover, we also assume that (during the stable period) all the content files served can be resided on the main memory and thus the delay caused by disk access is out of consideration. This is a reasonable assumption in many cases, since most current web servers have big main memory (> 128 MB) that is enough for most of the files. In our setup, each server has 256MB of main memory. Given the current average content file size, which is around 15K bytes, a set of 10,000 files occupies only 150MB. Other researchers make similar assumptions (see [11]).

2.2 Dispatcher-based Scheduling Algorithms

We will investigate the performance and scalability of the following scheduling algorithms on a dispatcher-based web cluster shown in Figure 3:

a. *Round-Robin (RR) Algorithm*
   In this scheme, the requests are dispatched to different servers in a round-robin manner. This scheme does not consider the current load of the servers.

b. *Least Connection (LC) Algorithm*
   The next request is assigned to the server with the least number of HTTP connections. This is a dynamic scheduling algorithm because it needs to keep a count of open connections for each server. This can be achieved by maintaining a server-connection table on the dispatcher.

c. *Least Loaded (baseline) Algorithm*
   In this scheme, the dispatcher assigns the next request to the server that has the lowest workload (workload of a server is defined as the sum of the service time of all requests pending on the server). The baseline algorithm requires knowledge about service time of the client requests. This information is often unknown as the requests arrive, and hence it is very difficult (if not impossible) to use the baseline algorithm in practice. However, one could use the baseline algorithm to establish an upper bound on the performance. Hence, we refer to it as the baseline algorithm and use it as a base for comparing with the performance of the other schemes.

3. Performance Modeling

There are different ways to study the performance of a system. Law and Kelton compares a number of scenarios such as experiment with the actual system versus experiment with a model of the system, physical model versus mathematical model, and analytical solution versus simulation [15]. However, some real systems, including the multiprocessor web-server we studied in this paper, are too complex to be evaluated analytically, i.e. to obtain closed-form solution, and the validation of analytical models is by itself a complex issue [15]. In computer simulation, we evaluate a model numerically, and data are gathered in order to estimate the desired true characteristics of the model. In this paper, we have adopted the trace-driven simulation approach to study the performance of a cluster-based web server. Trace log mimics real workload and can reproduce any effects due to a correlation between

different input parameters. An alternative is to use stochastic model of workload with appropriate probability distributions. An example of sucha model is the WEB-SPEC benchmark [22] but the need to include correlations among different parameters can lead to a workload model that is difficult to specify and to validate [5].

We developed a trace-driven discrete event simulator for the dispatcher-based system shown in Figure 3. The high-level structure of our simulator written in Java is shown in Figure 4. We focus on the behavior of two important entities in the server-cluster. One is the dispatcher and the other is the backend-web server. Each of these physical entities is abstracted as a Java class in the program. Trace parser is an interface class for handling different input trace formats. Dispatcher is a driver-class that received formatted requests from the trace parser and dispatches requests to one of the servers. It implements the load balancing algorithms and maintains a queue to hold pending requests. The server class when instantiated creates a number of objects that models the web servers.

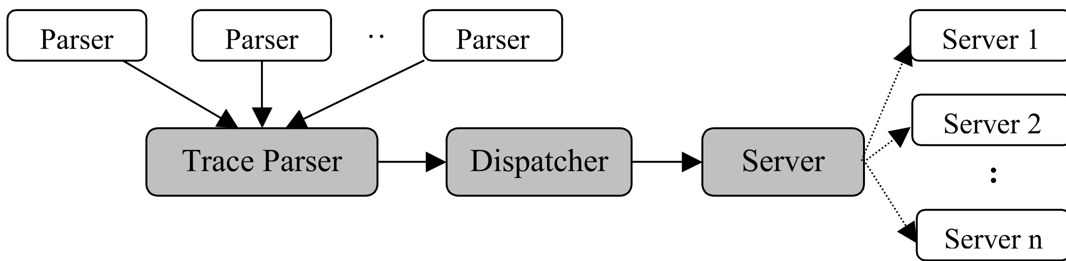

Figure 4: High-level Structure of Simulator

To increase the credibility and accuracy of our simulator, we first validate our simulator against simple analytical models and secondly with the result of our testbed consisting of 16 PCs. Since, scalability experiment with the testbed is limited by the size of the physical cluster, we use the validated simulator in our scalability studies.

3.1 Dispatcher Service Time

The dispatcher establishes one TCP session with each client after the client sends the request to the cluster. The TCP session ends when the client has successfully received the last reply packet sent by the back-end server. The dispatcher has to handle all the ACK (acknowledgement) packets sent by the clients during the TCP session since the clients know only the IP address of the dispatcher.

Each ACK packet is normally sent by the client in response to one reply data packet sent by the server. Therefore the larger the request size, the more reply data packets are sent by the server, which results in more ACK packets sent to the dispatcher by the clients. Thus the time the dispatcher needs to process each request is proportional to the size of the request. We hypothesize that

$$Dispatcher\ Service\ Time = L\ /\ K + C$$

L denotes the size of the request in bytes. K is a linear factor to model the processing of ACK packets received from clients, i.e., a client sends the dispatcher an ACK packet for every data packet it receives from the web server. C is a constant factor that models the overhead of setting up a TCP session.

Our simulation model mimics the test-bed we have in our parallel processing lab. The test-bed consists of a cluster of 16 homogeneous PCs interconnected by a 100 Mbps Ethernet. One of the PC acts as the dispatcher while the other 15 PCs are web servers. Each PC has a Pentium-II 400 Mhz processor, 256 MB of memory and runs Linux. The values of K and C are determine by carrying out some experiments on the testbed. This is done by sampling the request size from one byte to 2MB randomly, and measuring the average service time on the testbed. Figure 5 shows the dispatcher service time for varying request size. Based on our testbed hardware configuration, we observe that the dispatcher service time is a linear function with K = 1.4×10^{8} bytes/second and C = 120×10^{-6} second.

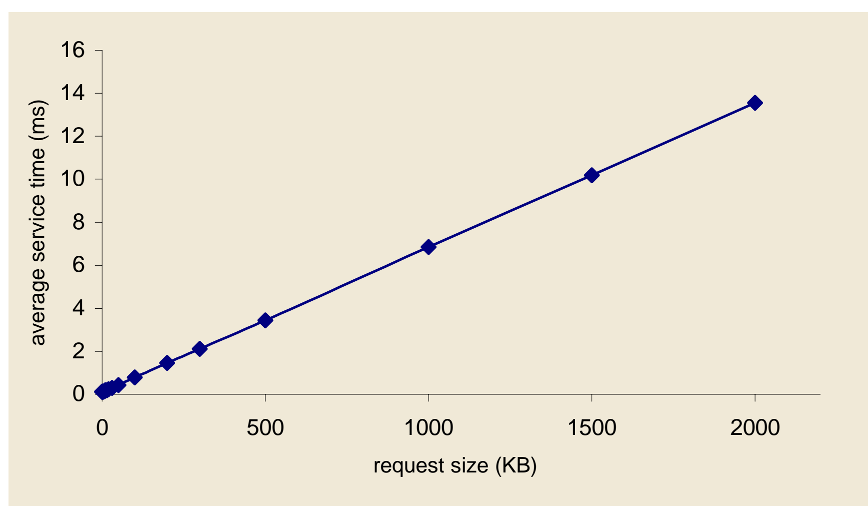

Figure 5: Dispatcher Service Time

3.2 Web Server Service Time

Compared to the dispatcher, the server model is more complicated because it involves nearly all components of a computer system. In a web server model there are three main delay sources that may affect the service time of a request. They are CPU (together with memory access) time, disk access time and network delay time. However, we may simplify the model. Firstly we observed that during system stable period, all the files of the server could be cached in the main memory. Thus the delay caused by disk access is not considered. Next we do not consider the LAN transmission delay between the dispatcher and the servers in our model. Based on these simplifications, we only consider the CPU processing time (plus memory access time) in servicing a request. Two classes of file transfers are considered: static and dynamic files:

a. Static File

This kind of requests does not need much computation and thus are characterized by fast CPU processing time. We hypothesize that:

$$Static\ File\ Service\ Time = L / K + C$$

We conducted several experiments, similar to the methodology used in determining dispatcher service time, on our test-bed and obtained the following values: K = 59.5 MB/second and C = 0.9 ms.

b. Dynamic File

This class of requests such as CGI files requires some computation work (i.e. a database query) with a spawned process or thread. Dynamic requests add serious burden to the CPU (a good discussion on the impact CGI-related requests on performance can be found in [11]). The service time for dynamic files depends on different applications on the server. Here we assume that the service time for dynamic files is a hundred times higher than the static files:

$$Dynamic\ File\ Service\ Time = L / K + C$$

We obtained the following values for these parameters on our test-bed: K = 0.595 MB/sec and C = 4.2 ms

4. Workload

In a trace-driven simulation, the selection of appropriate traces is a vital task. There are two main categories of traces: real and synthetic ones. A real web trace (available at public sites such as ACM SIGCOM or collected privately) reflects a sequence of real-life requests arrived at a specific web site at a specific time. On one hand such a trace is realistic and interesting, but on the other hand it may not be of general interest. In particular, such a trace may not be representative of the future web applications. Hence some researchers have developed tools for generating synthetic traces that mimic the main characteristic of the web traces (i.e., burstiness, self-similarity and heavy-tail distribution), but also offer the possibility to define your own parameters, such as arrival rate, request size, and inter-arrival time distribution. We conducted experiments using a synthetic trace and a realistic trace, as discussed below.

4.1 SURGE Synthetic Trace

We used the workload generation tool SURGE (Scalable URL Reference Generator) developed by Barford and Crovella at Boston University [4]. SURGE allows the user to specify almost all key aspects of web traffic, namely: document popularity, document size, temporal locality, spatial locality, off times of a single client, etc.

We generated a trace log file consisting of one million requests using SURGE. The selected workload parameters and other details are given in [7].

4.2 France Football World Cup 1998 Trace

We used a subset of the log file of the France Football World Cup's web site. The original file contains about 14 billion requests for a total period of 90 days. A detailed analysis of the trace log can be found in [17]. We used a one-day trace file (May 15th, 1998) that consists of 1,016,000 requests. By filtering invalid client request such as "Error 404: the file does not exist", the final trace file consists of about 900,000 requests.

Table 1 summarizes the main characteristics of both traces. The service time profile of HTTP requests shows that small requests dominate the web traffic. However, dynamic requests have longer service times. The bustiness factor is quantified using two parameters <a, b>, where *a* is the ratio between the above-average arrival rate and the average arrival rate for the entire trace period and *b* is defined as the fraction of time during which the epoch arrival rate exceeds the average arrival rate for the entire trace duration [18]. Generally, a bursty trace has a high a (≥ 3) and low b (≤ 0.3) and a non-bursty trace has a value close to 1 and 0.5 for *a* and *b* respectively. The burstiness factors presented in Table 1 are calculated by dividing the trace period into 100 epochs.

characteristics	**SURGE Trace**	**World Cup Trace**
total number of requests	1,060,043	902,433
trace period (seconds)	22,000	86,400
mean arrival rate (requests/second)	48.2	10.4
bustiness factor <a, b>	<1.53,0.48>	<1.58, 0.40>
mean request size (bytes)	13,564	8,237
total number of CGI requests	164,540 (16.4%)	95,675 (10.6%)

service time (ms)	**SURGE Trace (mean = 4.2 ms)**			**World Cup Trace (mean = 2.6 ms)**		
	total (%)	**static file**	**dynamic file**	**total (%)**	**static file**	**dynamic file**
> 500	2.9	0	2.9	1.1	0	1.1
50 – 500	8.2	0.3	7.9	4.4	0.5	3.9
5 – 49	19.7	15.4	4.3	11.1	6.7	4.4
< 5	70.2	69.9	0.3	83.3	82.2	1.1

Table 1: Characteristics of Trace Workload

5. Simulation Model Validation

We validated the simulator in two ways. First, we plugged in a request generator with Poisson arrival process and exponential service time distribution to the simulator and compared the result with the analytical ones, as explained below. Second, we run a set of web requests on our testbed and compared the obtained results with those of the simulator.

The request arrival process, together with the system model, can be viewed as M/M/1 (or M/M/c) queuing model. The following terms are commonly used in queuing theory:

λ = arrival rate (average number of customers arrived per time unit)

μ = service rate (average number of customers served per time unit)

ρ = utilization of the server

w = average waiting time of a customer in the queue

M/M/1 stands for a single server queue with Poisson arrivals having mean λ arrivals per time unit and exponential service times with mean μ^{-1} time units. Theoretically, we can get:

$$\rho = \lambda / \mu; \qquad w = \rho / (\lambda - \mu)$$

Table 2 illustrates the comparison of the analytical values of ρ and w with the simulation results ρ' and w'. We see that the simulation results are very close to the analytical ones.

λ	**M**	**Analytical Model**		**Simulation**	
		ρ = λ /μ	**w = ρ/(λ -μ)**	**ρ'**	**w'**
0.1	0.125	0.80	32	0.79	33.28
0.1	0.2	0.50	5	0.50	5.03
2	2.5	0.80	1.6	0.80	1.63

Table 2: Comparison of Simulator Results with M/M/1

Similarly, Table 3 compares the result of a multi-server system M/M/c with the result obtained by our simulator. As can be seen, the analytical results are very close to the results of the simulation.

no. of servers	**λ**	**μ**	**w (Analytic Model)**	**w' (Simulation)**
2	0.1	0.2	0.333	0.345
3	0.1	0.2	0.030	0.028
4	0.1	0.2	0.025	0.022
4	8	2.5	0.30	0.30

Table 3: Comparison of Simulator Results with M/M/c

We compare the simulator results with those obtained from the testbed in Table 4. The overall conclusions using both traces are similar, for brevity we present only the results from the Surge trace. In the testbed, one PC acts as the dispatcher, a few PCs are reserved to

generate real web requests, and the remainder models web servers. Thus, the scalability of our testbed is constrained by the size of our cluster of 16 PCs. In the validation experiment, we vary the number of web-servers from one to eight with both light and heavy load. Average response time is defined as the time interval between the arrival of a request and the end of service. The utilization of a server (U_i) is defined as the ratio between total amounts of time that server *i* is busy and the elapsed time. Average server utilization is defined as the sum of all server utilizations divided by the number of web servers.

no. of servers	arrival rate (requests/s)	average response time (ms)		average server utilization	
		testbed	simulation	testbed	simulation
1	100	31.7	27.9	0.394	0.362
Round Robin					
4	500	39.4	36.6	0.483	0.453
8	500	17.9	13.87	0.237	0.226
Least Connection					
4	500	10.2	7.5	0.481	0.453
8	500	11.7	3.9	0.232	0.226

Table 4: Surge Trace – Comparison of Simulator and Testbed Results

Table 4 shows that for average server utilization, the testbed results are higher than the simulator results but differs by less than 10%. The average server utilization in the testbed is derived by measuring the web-server CPU utilization, which includes the overhead of operating system processes. Average response times for the testbed and simulator are closer at low request arrival rate. For least connection, the different becomes larger because of the simplification we made in the simulator, i.e. scheduling overhead is negligible. Due to hardware constrain, the clients that generate requests share the same Ethernet switch with the web-servers in the testbed. Thus, as arrival rate increases the bottleneck effect of the outgoing link becomes prominent.

6. Performance Evaluation

To compare the relative performance of the three algorithms, we define the normalized waiting time and response time using a particular scheduling algorithm as:

i. **normalized waiting time** = average waiting time of an algorithm /
average waiting time of the *baseline* algorithm

ii. **normalized response time** = average response time of an algorithm /
average response time of the *baseline* algorithm

Since the baseline algorithm establishes the performance upper bound, a normalized time value of *one* denotes the best performance and a larger value denotes poorer performance. We conducted several experiments using the synthetic and the real traces, as explained below.

6.1 Experiments using SURGE Trace

In a cluster of computers, an important issue is the average utilization of the processors. Generally, if the cluster is highly overloaded no scheduling scheme can resolve the problem. On the other hand if the load is very low then almost any scheduling scheme would be acceptable and there is no need for sophisticated scheduling algorithms. Therefore, first we identify the saturation point of a single web server.

6.1.1 Capacity of a Single Web Server

The aim of this experiment is to identify the *average waiting time* and the *utilization* of a single server for various arrival rates. The average waiting time and server utilization are important measures for discussing load-balancing schemes.

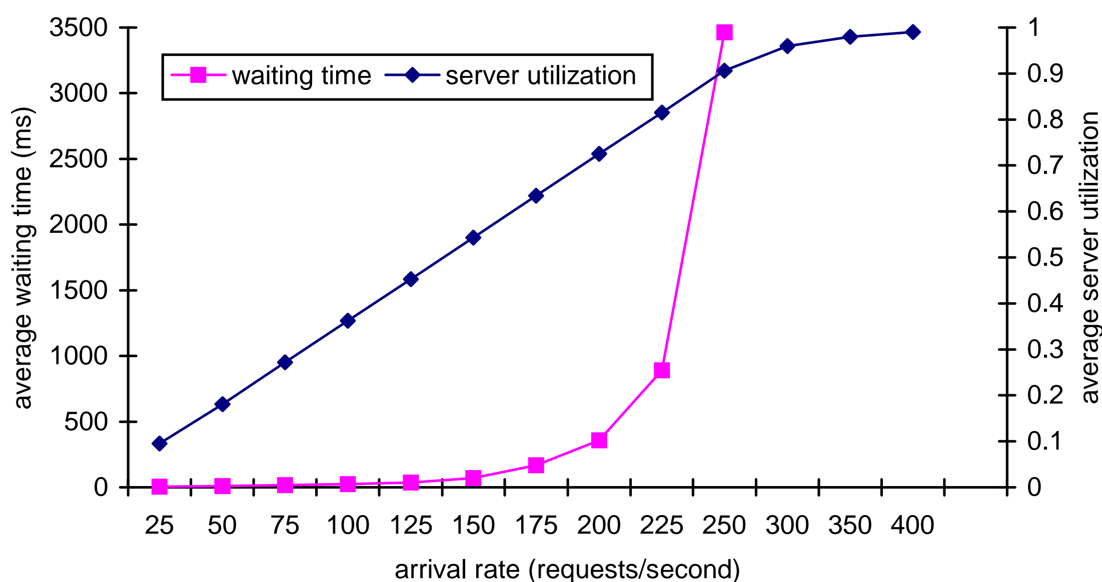

Figure 6: SURGE Trace - Capacity of a Single Web-Server

Figure 6 depicts the average waiting time and utilization of a single server system for various arrival rates (arrival rate is equal to the number of requests per time unit). As it can be seen, the average waiting time jumps from 800 to 3500 milliseconds when the arrival rate is increased from 225 to 250. In the utilization curve, when the arrival rate is 250 the utilization is close to 90%. This suggests that the saturation point of the single server is at 250 requests/second.

6.1.2 Multi-server Cluster

Figure 6 suggests that in a single server when the arrival rate is greater than 250 then the server is saturated and the queue length goes to infinity. We consider a web server with four

processors with a workload ranging from 250 to 1250 requests/second (i.e., 62.5 to 312.5 requests per server). We run the SURGE trace using three different scheduling algorithms, namely RR, LC and baseline. According to the results shown in table 3, the average waiting time using baseline algorithm is 853 milliseconds when the arrival rate is 1000 (which is equivalent to 250 requests/second to each of the four servers). The corresponding number (in figure 4) is 3500 milliseconds for the single server with arrival rate 250. Thus, the cluster provides a lower average waiting time than the single server with scaled down workload (i.e., 250 requests/second).

The "average server utilization" column in Table 5 illustrates the utilization of the servers and their deviation from the average value (denoted by ±) for the three algorithms. As it can be seen, the RR produces higher utilization deviation compared to the other two algorithms. The deviation could be seen as a measure for load imbalance.

arrival rate (requests/sec)	average waiting time (ms)			average response time (ms)			average server utilization		
	Baseline	RR	LC	Baseline	RR	LC	Baseline	RR	LC
250	0.2	10.5	0.9	3.8	14.1	4.5	0.226± 0.001	0.226± 0.002	0.23± 0.001
500	1.8	33.0	3.9	5.4	36.6	7.5	0.453± 0.002	0.453± 0.003	0.453± 0.002
750	49.5	127.5	53.1	53.2	131.1	56.7	0.679± 0.001	0.679± 0.004	0.680± 0.001
1000	849.5	1112.3	853.0	853.1	1115.9	856.7	0.906± 0.00	0.905± 0.006	0.905± 0.001
1250	70084	70118	70085	70087	70121	70088	0.998± 0.001	0.991± 0.006	0.997± 0.001

Table 5: Performance of a 4-Server Cluster

The normalized waiting time in Figure 7 shows that the performance gap between the three algorithms is wide for low arrival rates. For instance, with 250 requests/second the RR performs fifty-two times worse than the baseline algorithm whereas the LC performs five times worse. However, the normalized average waiting times of the two algorithms gradually converge to the baseline as the arrival rates (workload) increases. It is clear that the LC is much better than the RR as LC performs much closer to the baseline (with normalized waiting time close to *one*).

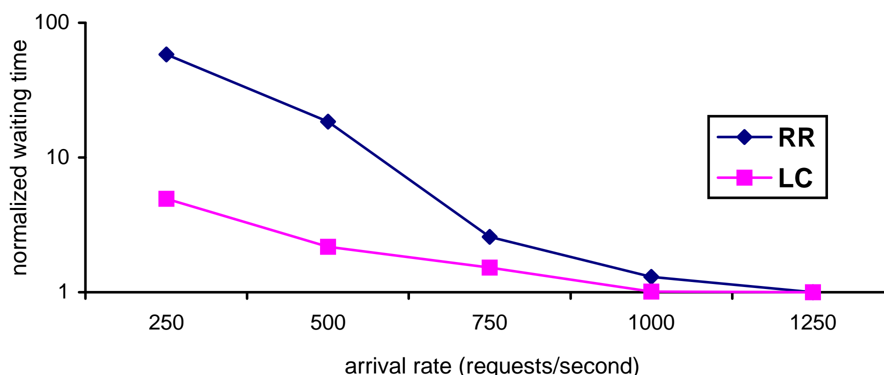

Figure 7: 4-Server Cluster – Normalized Waiting Time

6.1.3 Scalability Experiments

First we scale up the number of computers and the workload proportionally (i.e., the workload of each server remains constant). We run a set of experiments assuming that each server of the cluster is as powerful as the single server. However, we increase the number of servers and the workload of the cluster proportionally, i.e., by increasing the request arrival rate.

Table 6 illustrates the average waiting time, average response time, and average server utilization for various number of servers with a constant load per server. We observe that with the growth in the number of servers, the average waiting time keeps declining.

no. of servers	arrival rate	average response time (ms)			average waiting time (ms)			average server utilization		
		baseline	RR	LC	baseline	RR	LC	baseline	RR	LC
1	250	3467			3467			0.906		
2	500	1722.0	1913.2	1724.6	1718.4	1909.5	1721.0	0.906 ±0.000	0.905 ±0.002	0.905 ±0.000
4	1000	853.1	1115.9	856.7	849.5	1112.3	853.0	0.906± 0.00	0.905± 0.006	0.905± 0.001
8	2000	419.7	741.4	421.6	416.0	737.8	418.0	0.906 ±0.001	0.905 ±0.007	0.906 ±0.001
16	4000	213.0	608.0	215.6	209.4	604.4	212.0	0.906 ±0.001	0.903 ±0.017	0.905 ±0.002

Table 6: Varying Number of Servers with the Power of Server Remains Constant

Figure 8 depicts the normalized waiting time for various numbers of servers. We observe that the normalized waiting time of RR is increased when the number of serves grows, whereas the LC performs constantly very close to the baseline algorithm (it performs at most with 5% worse for 16 servers).

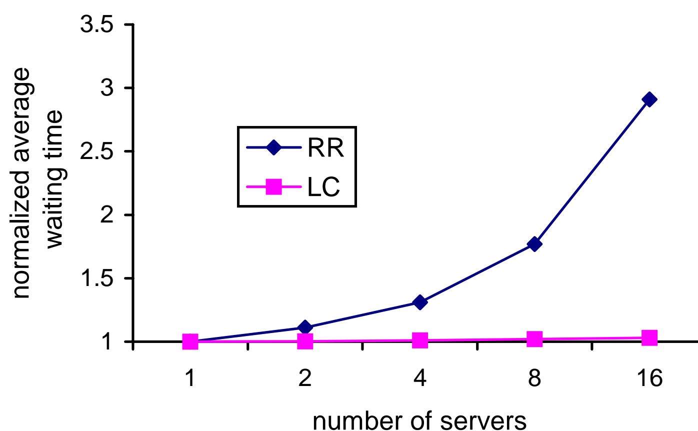

Figure 8: Normalized Waiting Time (Surge Trace)

In the second set of scalability experiment we scale up the number of servers while the computation power of each server is decreased proportionally. We run a set of experiments on a cluster with n servers where the computation power of the single server is X and the

power of each server is equal to X/n. Consequently the computation power of the cluster and its workload is equal to the single server. Practically, we replace the single server with n servers, each having power X/n. The performance results shown in Table 7 indicate that the choice of scheduling algorithms is essential. If RR is used the cluster produces a much higher average waiting time and response time than the single server, when the number of servers grows. On the other hand, we see that with good distribution strategies like LC and baseline, the cluster produces a better performance than the single server.

As for the relative performance, again we can see from Figure 9 that RR cannot cope with the growth in the number of servers whereas the LC stays very close to the baseline algorithm (within 5%). Column 1 in Table 7 shows the number of servers and the power of each server (given in parenthesis)

no. of servers	average waiting time			average response time			average server utilization		
	baseline	RR	LC	baseline	RR	LC	baseline	RR	LC
1(X)	3463			3467			0.906		
2(X/2)	3436	3818	3445	3444	3826	3452	0.906± 0.000	0.905± 0.002	0.905± 0.000
4(X/4)	3397	4447	3409	3412	4461	3423	0.906± 0.000	0.905± 0.006	0.906± 0.001
8(X/8)	3326	5899	3350	3355	5928	3379	0.906± 0.001	0.904± 0.007	0.906± 0.001
16(X/16)	3239	8730	3267	3297	8788	3365	0.906± 0.002	0.903± 0.017	0.905± 0.002
32(X/32)	3075	14428	3146	3191	14544	3262	0.905± 0.003	0.900± 0.024	0.904± 0.002

Table 7: Varying Number of Servers with a Fixed Workload

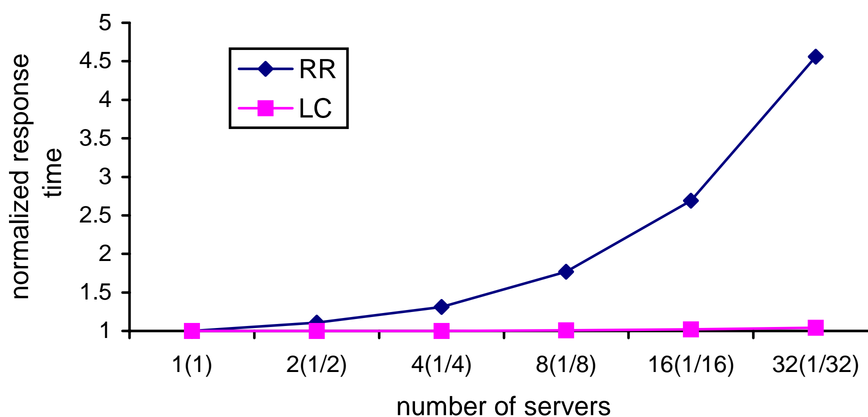

Figure 9: Normalized Response Time (Surge Trace)

6.2 Experiments using the Football World Cup 1998 Trace

6.2.1 Capacity of a Single Web Server

In this experiment, we use the 1998 Football World Cup trace, but scale up the arrival rate of the requests (by multiplying the inter-arrival times by a fixed factor).

Figure 10 depicts the average waiting time and utilization of a single server system for the World Cup Trace. As it can be seen, the saturation point of the single server for this trace is at 350 requests/second, which is higher than the 250 requests/second of the SURGE trace.

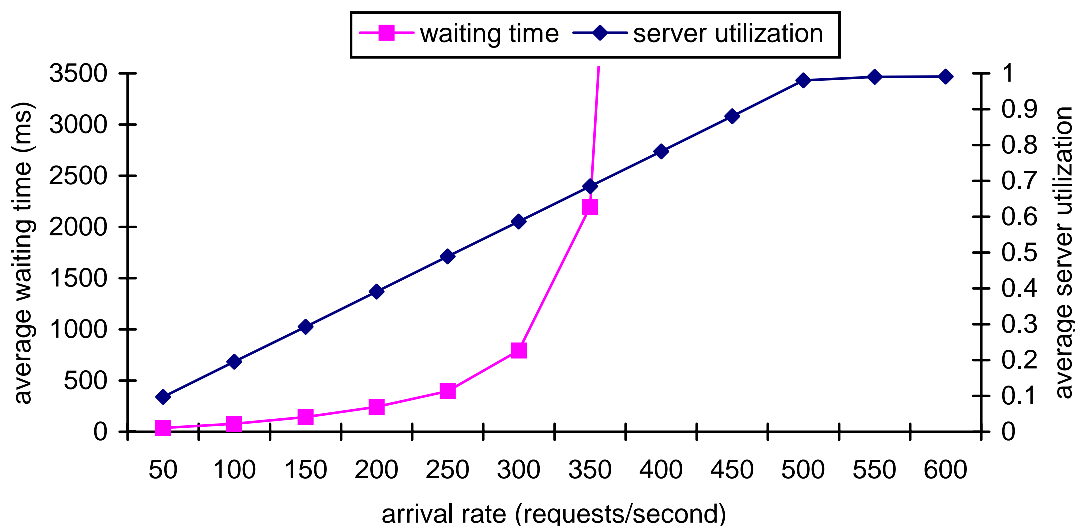

Figure 10: Capacity of a Single Server - World Cup Trace

6.2.2 Multi-server Cluster

The normalized waiting times of the three scheduling algorithms for the World Cup trace (on a 4 processor cluster) is shown in Figure 11. The figure confirms that for low workload the performance gap between the three algorithms are wide, while with increasing workload the differences between the waiting times for the three algorithms are reduced.

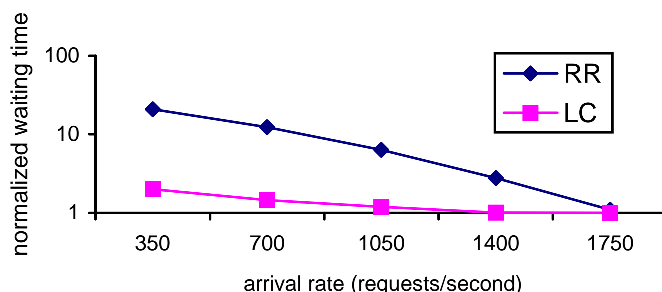

Figure 11: 4-Server Cluster - Normalized Waiting Time

6.2.3 Scalability Experiments

We run a set of experiments, where the number of servers and the workload are scaled up (similar to those run on the SURGE trace). Figure 12 depicts the normalized waiting time for up to 8 servers. Here, we also observe that the normalized average waiting time of the RR is increased when the number of serves grows. The LC performs constantly very close to the baseline algorithm.

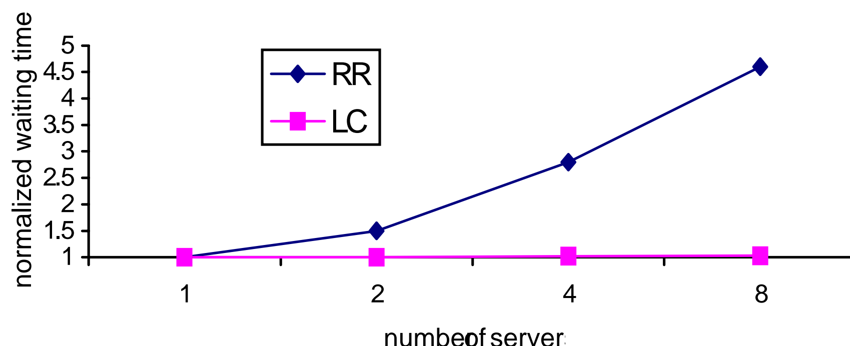

Figure 12: Normalized Waiting Time (World Cup Trace)

The second set of scalability experiments is run on a set of less powerful servers, where the workload remains constant, the number of servers are increased, and the computation power of each server is scaled down (similar to those run on the SURGE trace). The performance results shown in Figure 13 confirm that RR scheduling cannot cope with the growth in the number of servers whereas the LC stays very close to the baseline algorithm.

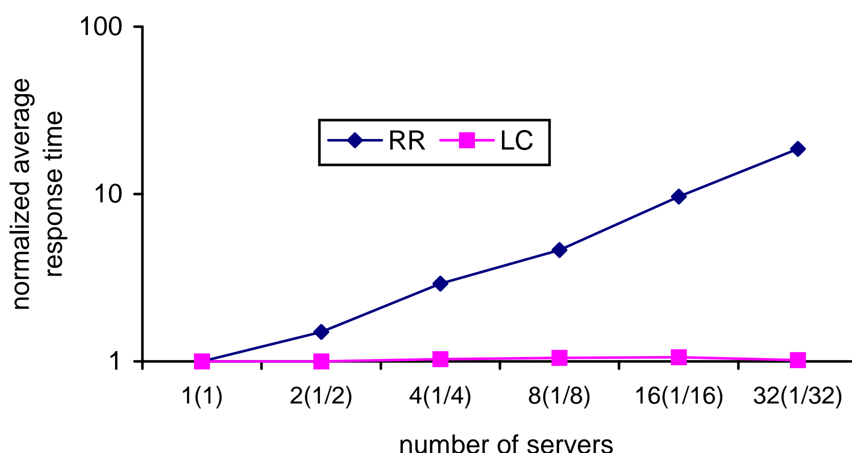

Figure 13: Normalized Response Time (World Cup Trace)

7. Summary and Conclusions

We developed two experimental platforms, a web-server cluster consisting of 16 PCs and a simulator, to investigate load balancing and scalability issues in cluster-based web servers. The service times of the dispatcher and the servers are measured and used as input to the simulator. The PC cluster produces more realistic results, but it cannot be used for scalability studies since it would require the purchase of additional processors and faster Ethernet Card. On the other hand, it is easier to modify the parameters of the simulator and conduct scalability studies. Hence, we used both of these platforms in our study.

The following simplifying assumptions have been made in the simulator

- The scheduling overhead, i.e., the time needed to assign the next task to a processor, is negligible;
- The communication overhead, i.e., the time needed to send a message from the dispatcher to a server, is negligible;
- The bandwidth of the outgoing link is unlimited (or sufficient to handle the outgoing traffic).

At an early stage, our experiments on the cluster revealed that the outgoing link capacity would be a bottleneck when the number of servers exceeds 8. However, our focus was not the outgoing network bandwidth and hence we assumed unlimited outgoing link capacity for the simulator.

As input to our simulator, we used the World Cup trace (a publicly available log-file) and synthetic traces generated by SURGE. We identified the limit of a single server and used it in configuring our simulator and investigated the performance of three load scheduling algorithms, namely least loaded first (denoted as baseline), least connection first (LC), and round robin (RR). Our investigation shows that:

- The baseline algorithm performs best, but it needs information about service time requirement of each request. This information is usually unavailable in realistic simulations and thus it is difficult to employ the baseline algorithm.
- Round Robin performs much worse than the other two algorithms for low to medium workload. However, when the arrival rate to the cluster increases, the performances of the three algorithms start to converge. The performance of the Least Connection approaches the baseline algorithm in a much faster speed than Round Robin.
- The least connection algorithm is easy to implement and it performs well for medium to high workloads. However, when the workload is very low, the waiting time of the Least Connection scheduling is considerably higher than the baseline algorithm (2-6 times higher). But, for such low workloads the absolute gap between these two waiting times is very low and thus it may still fulfil the response time (or deadline) required by the end user.

Acknowledgement

This project is supported by a joint research grant from Fujitsu Computers (Pte) Ltd and the National University of Singapore. The authors would like to thank Chen Ting for his help in this project.

References

1. D. Anderson et al., "SWEB: Toward a Scalable World Wide Web-Server on Multicomputers," *Proceedings of 10th IEEE International Symposium on Parallel Processing*, pp. 850-856, 1996.

2. E. Anderson, D. Patterson, and E. Brewer, "The Magicrouter, an Application of Fast Packet Interposing," *Proceedings of 2nd Symposium on Operating System Design and Implementation*, 1996.

3. L. Aversa and A. Bestavros, "Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting," *Proceedings of IEEE International Performance, Computing and Communication Conference*, Phoenix, USA, February 2000.

4. P. Barford and M. Crovella, "Generating Representative Web Workload for Network and Server Performance Evaluation," Technical Report 1997-006, Computer Science Department, Boston University, 1997.

5. H. Bryhni, E. Klovning and O. Kure, "A Comparison of Load Balancing Techniques for Scalable Web Servers," pp. 58-63, IEEE Network, July/August 2000.

6. V. Cardellini, M. Colajanni and P.S. Yu, "Dynamic Load Balancing on Web-Server Systems," *IEEE Internet Computing*, pp. 28-39, May-June, 1999.

7. T. Chen, "Load Balancing Strategies on Cluster-based Web Servers," Project Report, Department of Computing Science, National University of Singapore, http://www.comp.nus.edu.sg/~teoym/ref7.pdf, 2000.

8. Cisco Local Director, http://www.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml.

9. D.M. Dias et al., "A Scalable and Highly Available Web Server," Proceedings of IEEE COMPCON, 1996.

10. P. Damani, P. Chung, Y. Huang, C. Kintala and Y. M. Wang, "ONE-IP: Techniques for hosting a service on a cluster of machines," *Computer Networks and ISDN systems*, Vol. 29, pp. 1019-1027, 1997.

11. Y. Hu, Nanda, A. and Q. Yang, "Measurement, Analysis and Performance Improvement of the Apache Web Server Performance," *Proceedings of IEEE Computing and Communications Conference*, pp. 261 –267, 1999.

12. G. Hunt, "Network Dispatcher: a connection router for scalable Internet services," *Computer Networks and ISDN Systems,* 30(1998), pp. 347-357.

13. A. Iyengar, A. MacNair and E. Nguyen, "An Analysis of Web Server Performance," *IEEE GLOBECOM '97, Volume: 3,* pp. 1943 –1947, 1997.

14. T.T. Kwan, R.E. McGrath, and D.A. Reed, "NCSA's World Wide Web server: Design and Performance", *IEEE Computer*, pp. 68-74, Nov.1995.

15. A.M. Law and W.D. Kelton, "Simulation Modeling and Analysis," 3rd edition, *McGraw Hill*, 2000.

16. Linux Virtual Server Project, http://www.linuxvirtualserver.org/scheduling.html.

17. A. Martin and J. Tai, *"Workload Characterization of the 1998 World Cup Web Site,"* http:// www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html.

18. D.A. Menasce and V.F. Almeida, "Capacity Planning for Web Performance," *Prentice-Hall*, 1998.

19. J. Mogul, "Network Behavior of a Busy Web Server and its Clients," Research Report 95/5, DEC Western Research Laboratory, October 1995.

20. V. Pai et al., "Locality-Aware Request Distribution in Cluster-based Network Servers," *Proceedings of ACM 8th Int'l. Conf. Architectural Support for Prog. Langs. and Op. Sys.*, October 1998.

21. T. Schroeder, S. Goddard and B. Ramamurthy, "Scalable Web Server Clustering Technologies," pp. 38-45, IEEE Network, May/June 2000.

22. Standard Performance Evaluation Corp. (SPEC), SPECWeb99 Benchmark, http://www.specbench.org/osg/Web99, 1999.