

PAPER

# Dynamic Mirroring for Efficient Web Server Performance Management

Shadan SANIEPOUR E.<sup>†</sup>, Behrouz Homayoun FAR<sup>††</sup>, *Regular Members,*  
and Jingde CHENG<sup>†</sup>, *Nonmember*

**SUMMARY** Server performance is a major issue in improving the overall performance of the World Wide Web (WWW). This article introduces a dynamic mirroring-based approach to improve WWW servers' performance. In contrast to static mirroring, where mirror servers are allocated statically, our mirror servers' setup is driven by network traffic measurement. Performance in terms of latency is inferred from a queuing model. According to this model we show that latency of an overloaded server can be tuned by delegating a portion of the load to a cooperative mirror server. Cost is evaluated by the amount of load hosted by the mirror servers. The goal is then to keep the latency within a tolerable threshold, while minimizing the delegated load. This problem is formulated as a constrained optimization problem where the task is to assign a portion of load corresponding to each document to each mirror server. As the result of this work, we will have a balanced load among the servers, and a smoother traffic along the Internet, as well. Empirical results show that this approach can guarantee to maintain the performance while showing a significant decrease in the amount of load transferred to the mirror servers.

**key words:** *web server, performance, mirroring, traffic, queuing theory, optimization*

## 1. Introduction

The explosive growth of the World Wide Web traffic in the recent years has made the Web the largest contributor to traffic of the Internet. Long latency perceived by the clients, caused by disproportionate traffic on the web servers, has been the main motivation for studies on web workload characterization and web traffic engineering. Proxy caching and server mirroring are the proposed solutions to control the traffic.

The rationale for server mirroring is to distribute the traffic of a web server among several peers. However maintaining several servers in different geographical locations, raises economical concern for small organizations. In addition, providing an efficient traffic routing strategy, to distribute the traffic properly, is an issue [1], [4], [8], [10], [12].

Caching protects the web servers from rush of access by transferring data to Proxy servers or local caches [2], [5], [7], [27], [28], [30]. As the result it reduces

the traffic of the web servers. In order to gain a good performance, a cache maintainer (i.e., proxy server or the local cache) should dynamically manage the cache: refresh the cache or decide which document to cache. Furthermore, accessing data via several proxy servers requires an efficient cache hitting algorithm.

The ultimate goal of each of these approaches is to reduce the latency by means of controlling the traffic. We explicitly define our goal as maintaining a reasonable average latency for clients, and study the web servers performance management and traffic control from this viewpoint. We account for the cost of performance management and make a tradeoff between client perceived latency and the cost of performance management. To manage the performance of a web server, the most accurate statistics about the load characteristic can be derived from the web server. Therefore we choose to apply performance management in the server side, as opposed to proxy caching and pre-fetching which are applied in the client side. Analyzing the previous approaches suggests that, in general, the main problem with mirroring is the cost and traffic distribution. In caching, several servers share the same auxiliary servers, which are proxy servers, or local caches that perform like local servers. Therefore cost is not a problem. However caching is absolutely arbitrary from server viewpoint and there is no principled way of controlling web server performance via caching.

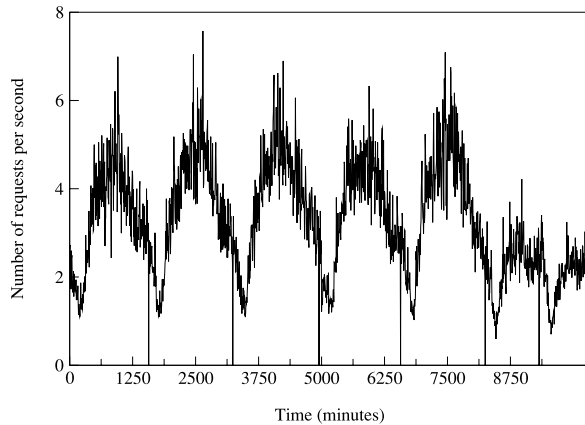
Recent studies on traffic issues related to web servers have shown distinct patterns in the amount of requests received by a web server, depending on the time of a day: peak time during busy hours, and the time that web server is lightly loaded [5], [13], [22]. Figure 1 shows the amount of request per second for a typical busy server during one week, extracted from [6]. This phenomenon suggests the possibility of using mirroring services only during the peak time. In this work we introduce the notion of temporal and partial mirroring to formulate this intuition. This is to alleviate the cost of mirroring by sharing mirror servers. Mirroring is applied only if a server fails to maintain the desired performance. Performance is evaluated in terms of latency. If a server fails to provide a reasonable expected latency, most of the clients are likely to abandon retrieving data. This situation is referred to as performance deficiency, and a server that demonstrates

Manuscript received September 13, 2001.

Manuscript revised January 10, 2002.

<sup>†</sup>The authors are with the Department of Information and Computer Science, Saitama University, Saitama-shi, 338-8570 Japan.

<sup>††</sup>The author is with the Department of Electrical & Computer Engineering, Calgary University, Canada.



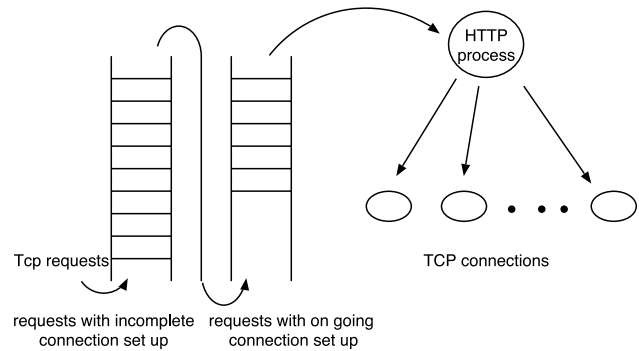
**Fig. 1** Traces of traffic of a busy server (Clarknet Server) for duration of one week.

performance deficiency is referred to as an overloaded server. An overloaded server lifts a portion of its load, by mirroring a part of its documents on another server. Mirroring continues, for as long as the overloaded server fails to serve its total load with the desired performance.

In practice this approach can be realized by allocating rental mirror servers. As the mirroring cost is evaluated by the amount of load, hosted by each mirror server, web servers' goal is to transfer as much load to maintain the performance, but as little load to receive the minimum charge.

In our approach, web servers are modeled as queuing systems and their latency, as well as inter-arrival rate of the requests, service rate of TCP connections, and average retrieved document size is dynamically monitored. As soon as performance deficiency is anticipated, mirroring is applied to lift a portion of the load. In this regard, traffic situation of each mirror server is enquired, to help selecting the best cooperative mirroring partner(s). Also, analyzing the traffic generated by different documents enables us to decide upon assigning a portion of traffic corresponding to which document to which mirror server. Ultimately a copy of selected documents are sent to the selected mirroring partners and the decided proportion of their generated load is transferred to the new destinations. Mirroring continues for as long as the server fails to accommodate its total load while maintaining the performance.

The rest of this paper is organized as follows. In the next section we study the issues related to performance of web servers and propose a queuing system to model a web server. In Sect.3 we discuss how to tune the parameters of the derived model, in order to maintain the performance. In Sect.4, we estimate the performance of an overloaded server if a portion of load is lifted from it, and performance of a mirror server if an extra amount of load is assigned to it. Section 5 describes how the decision is made to assign a portion of traffic generated by each document to each mirror



**Fig. 2** HTTP receives the requests queued by TCP, and serve them via multiple TCP connections.

server, in order to achieve the best performance along with the minimal cost. Experimental results are presented in Sect.6. Finally we conclude in Sect.7.

## 2. Performance Monitoring

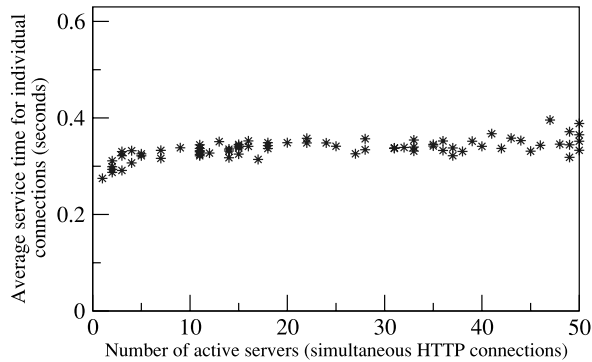
The goal of performance monitoring is to monitor the average response time. Average response time is the time interval which on average takes for the web server to serve a request. A web server uses HTTP protocol, implemented over TCP/IP, for data transfer. Therefore analyzing the performance of a web server is correlated with the behavior of TCP. Figure 2 illustrates, how requests are handled by the HTTP process.

As illustrated in Fig.2, TCP process buffers all incoming requests before passing them to HTTP process. HTTP process then, establishes a TCP connection to serve each request. Concurrent servers refer to the web servers that allow establishing multiple simultaneous TCP connections, each serving an HTTP request. The lifetime of each connection is correlated with the throughput of TCP connections, and the size of the requested document.

As the requests are queued before processing, the behavior of a web server can be modeled as a queuing system. Expected latency can be inferred from this model. Similar formulation is proposed to model a multiprocessor environment [17], [25], where the goal is to maintain the shortest queue for all the processors. We are however interested in the average latency of the servers. The next concern is to define the parameters of the queuing system that can properly model our system.

### 2.1 The Queuing Model

A queuing system is identified by: distribution of the inter-arrival time of the requests, distribution of the service time, the queuing discipline (FIFO, PS, etc), the maximum buffer size and number of servers in case of multi-server system. The inter-arrival time is denoted as  $a$  and service time is denoted as  $s$ , the arrival rate is



**Fig. 3** Changes in service time when number of simultaneous HTTP jobs increases. This experiment is conducted in simulated network using dummynet, RTT=80, BW= 400 Kb/s, 600 Kb/s, 1 Mb/s,  $\bar{D} = 7$  kbytes.

then given by  $\lambda = 1/E(a)$ , and the service rate is given by  $\mu = 1/E(s)$ . For a queuing system with  $c$  servers, the utilization  $\rho = \frac{\lambda}{c\mu}$ , is known as the mean fraction of active servers [3].

Studies on traffic engineering and web server workload have shown that distribution of inter-arrival times and web servers response times can be modeled as Weibull distribution [14], [18]. We therefore analyze the system under general case, considering general distributions for inter-arrival times and servers response times.

Since all the jobs are ultimately processed by one CPU, it is expected to model the system as queuing system with PS (Processor Sharing) discipline, to account for decreasing the service rate as the number of jobs grow in the system. However as shown in Fig. 3, because of the particular characteristic of web servers resulted from the nature of the TCP process, there is no significant change in the amount of average service time, as number of simultaneous jobs grow. This phenomenon is mainly resulted by discrete nature of TCP jobs. TCP process sends a window of segments and waits for the acknowledgments and over receiving ACK (or timeouts) sends the new segments. This process takes at least RTT (Round Trip Time), which is relatively large<sup>†</sup>. From TCP point of view CPU becomes idle after sending a window and can start serving a new connection. In addition, a web server imposes some limitation on the number of simultaneous servers. In a well tuned system, a reasonable number of simultaneous servers, protects the system against long latencies. As the blocking time for TCP jobs is relatively large (i.e., at least RTT) and the number of concurrent connections is limited, the processor sharing discipline is not severe in the system, and the system can be approximated as a multi-server queuing system, with FIFO discipline. Note that this approximation simplifies further approximations for estimating response time of the system.

Therefore our queuing model follows a general form of  $GI/G/c/k$  model, where  $c$  denotes the total number

of servers (i.e., maximum number of connections created by HTTP process), and  $k$  defines the maximum buffer size. Since the buffer size is relatively large<sup>††</sup>, we approximate web servers as queuing systems with infinite buffer size.

In practice  $\lambda$  increases gradually when we get closer to the peak time, and decreases gradually around less busier hours. This implies the fact that the sampled inter-arrival time is shifting to a new distribution. We dynamically monitor the mean and variance of inter-arrivals and service time, in order to keep trace of changes in the data. As we show later, these are the parameters that allow estimating the performance.

## 2.2 Estimating Response Time

The expected waiting time in the system is:

$$W = W_Q + 1/\mu \quad (1)$$

where  $W_Q$  denotes expected waiting time in the queue.

Following approximation derived from Eq. (6.94) of [3], allows estimating  $W_Q$  even if the number of servers is large.

$$W_Q = \frac{\rho/\mu}{1-\rho} \frac{cv_a^{f(cv_a, cv_s, \rho)} + cv_s^2}{2} \rho^{0.25(c-1)} \quad (2)$$

$$f(cv_a, cv_s, \rho) = \begin{cases} 1 & cv_a \in \{0, 1\} \\ (\rho(14.1cv_a - 5.9) + (-13.7cv_a + 4.1))cv_s^2 \\ + (\rho(-59.7cv_a + 21.1) + (54.9 - 16.3))cv_s \\ + (\rho(cv_a - 4.5) + (-1.5cv_a + 6.55)) & 0 < cv_a < 1 \\ -0.75\rho + 2.775 & cv_a > 1 \end{cases} \quad (3)$$

where  $cv_a$  and  $cv_s$  are coefficient of variation of inter-arrival times and service times respectively. The goal is to keep the response time of the system under a desirable threshold denoted as  $T_0$ . Whenever response time of the system derived from (1) is estimated more than  $T_0$ , the system is experiencing overload. As can be concluded from (2) waiting time in the system is a function of  $\rho$ ,  $cv_a$  and  $cv_s$ . Therefore in order to tune the waiting time we can adjust these parameters. In the next section we study how these parameters change when we transfer a portion of the load to mirror servers.

## 3. Impact of Load Transfer on Parameters of the Queuing Model

As described earlier, in order to maintain the perfor-

<sup>†</sup>A usual value of 75 msec is commonly used in different simulations.

<sup>††</sup>In Solaris as default, maximum buffer size for connections with incomplete setup is (tcp.conn.req.max.q0 = 1024), and Maximum buffer size for the connection with ongoing connection setup is (tcp.conn.req.max.q = 128).

mance for an overloaded server, our proposal is to transfer a portion of the load to the mirror servers. Load transferring approaches can be categorized in two main classes: approaches in which overloaded server is responsible for transferring the load (i.e., redirection) and, approaches in which load transfer is applied via a separated device (e.g., Layer 7 switching). We refer to the former as redirection and latter as switching. The main difference between two approaches is in the value of arrival rate. In case of redirection all the requests arrive at the overloaded server and then are redirected, which leaves the arrival rate unchanged. In the other hand, as given by (4) using switching reduces the arrival rate of the overloaded server as the amount of transferred load increases.

$$\lambda_{new} = \begin{cases} \lambda & \text{redirection} \\ \lambda - \lambda^{transferred} & \text{switching} \end{cases} \quad (4)$$

where  $\lambda_{new}$  denotes the new arrival rate after applying load balancing, and  $\lambda^{transferred}$  denotes the total request rate forwarded to mirror servers. Finally, decreasing arrival rate ( $\lambda$ ) results in decreasing  $\rho$  and enable us to control latency.

Regardless of the applied load transfer approach, by transferring a portion of the load we can adjust the average size of the transferred documents. We show that  $\mu$  has an inverse proportion with average transferred document size. Therefore by altering average document size we can tune the  $\mu$  and consequently  $\rho$ , and ultimately we can adjust latency.

If the connection throughput of each TCP connection is denoted as  $CT$ , the service time of a TCP connection  $s$  to serve the document  $D$  satisfies:

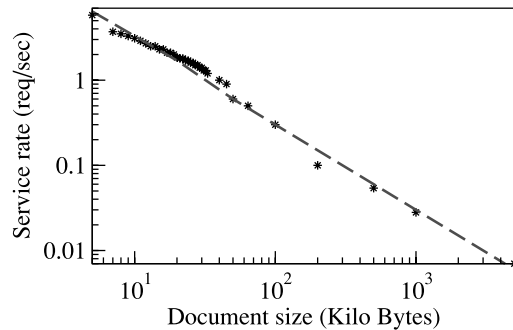
$$D = \int_0^s CT(t) dt \quad (5)$$

$$\begin{aligned} E(D) &= E\left(\int_0^s CT(t) dt\right) \\ &= E_s\left(E\left(\int_0^x CT(t) dt | s = x\right)\right) \\ &= E_s(sE(CT)) \\ &= E(s)E(CT) \end{aligned}$$

and finally we will have:

$$\mu = \frac{1}{E(s)} = \frac{E(CT)}{\bar{D}} \quad (6)$$

$\bar{D}$  indicates the average document size, in KB. Note that  $CT$  is assumed to be stationary. Equation (6) implies that  $\mu$  can be tuned by adjusting  $\bar{D}$  or  $CT$  independently. However, it has been shown that  $CT$  is a function of  $RTT$ , packet loss and other limitations imposed by the client (maximum window size) and adjusting it is not under our control [9], [16], [19]. Our



**Fig. 4** Relation between service rate and average document size plotted in logarithmic scale, which shows  $\log(\mu) = -\log(\bar{D}) + \{\log(\mu_0) + \log(\bar{D}_0)\}$ .

objective is to tune the  $\mu$  by altering  $\bar{D}$ . Following equation shows how  $\mu$  is changed if we alter average document size.

$$\mu_{new} = \mu \cdot \bar{D} / \bar{D}_{new} \quad (7)$$

where  $\mu_{new}$  and  $\bar{D}_{new}$  denote the values after change. The verification of this equation is illustrated in Fig. 4.

The average document size is given by:

$$\bar{D} = \frac{\sum_i n_i d_i}{N} \quad (8)$$

where  $N$  denotes total number of requests received at this server,  $n_i$  denotes the number of requests corresponding to document  $i$ , and  $d_i$  denotes the size of document  $i$  in kilo bytes. Considering the fact that  $n_i = \hat{q}_i \cdot t$  where  $\hat{q}_i$  denotes the total request rate corresponding to document  $i$ , and  $N = \lambda \cdot t$ , Eq. (8) is rewritten in (9) to indicate the relation between  $\bar{D}$  and request rate.

$$\bar{D} = \frac{\sum_i \hat{q}_i d_i}{\lambda} \quad (9)$$

According to (9) reducing average document size is possible via reducing the request rate corresponding to some of the documents, which is possible by transferring part of the requests to a mirror server.

To estimate the coefficient of variation of service time after transferring the load, with the similar reasoning as shown in (5)–(7) it can be shown that:

$$(cv_s^{new})^2 = \frac{\bar{D}^{2new} / (\bar{D}^{new})^2}{\bar{D}^2 / (\bar{D})^2} \cdot (cv_s^2 + 1) - 1 \quad (10)$$

In order to estimate the coefficient of variation of the inter-arrival time after transferring a portion of the load, we need to decompose the arrival stream to its corresponding streams of requests for different documents. Given that we know the rate of changes in the request rate of different documents we can estimate the total change in the coefficient of variation of the inter-arrival time. If the inter-arrival time of

the requests corresponding to different documents are batch-Poisson and Batch-deterministic process with geometric batch sizes, the total coefficient of variation of the inter-arrival times according to (8) of [29] can be derived from (11). Note that assumption of geometric batch sizes makes the individual clients inter-arrival time independent and identically distributed (GI process) [29].

$$\begin{aligned} cv_a^2 &= \sum_{i=1}^K cv_i^2 \frac{n_i^{new}}{N} \\ &= \sum_{i=1}^K cv_i^2 \frac{\hat{q}_i^{new}}{\lambda} \end{aligned} \quad (11)$$

where  $cv_i$  denotes the coefficient of variation for the inter-arrival time of the requests for document  $i$  and  $n_i^{new}$  and  $\hat{q}_i^{new}$  respectively denote the number of requests corresponding to document  $i$  and the request rate corresponding to document  $i$ .

In this section we studied the impact of load transfer on tuning  $\lambda$ ,  $\mu$ ,  $\rho$ ,  $cv_s$  and  $cv_a$ . In the next section we describe how to estimate the latency after tuning these parameters.

#### 4. Estimating Performance Parameters after Load Sharing

Transferring a portion of the load to a mirror server, alters the performance of both parties, the server that transfers its load and the mirror server that receives extra load. In this section we estimate the latency for the overloaded server and mirror servers after transferring the load. To clarify the terminology, subscript  $j$  is used to specify the parameters of mirror servers, whereas subscript  $o$  specifies the parameters of the overloaded server. The integer  $M$  denotes the number of mirror servers, as the integer  $K$  denotes the number of documents on the overloaded server. Each mirror server is denoted as  $mirror_j$  where  $j = 1, 2, \dots, M$ . The request rate corresponding to document  $i$  which is assigned to the mirror server  $j$  is denoted as  $q_{ij}$ .

To estimate the new average document size (9) can be used. If load is transferred via redirection the average arrival rate remains unchanged. The new average document size in this case is derived from (12).

$$\bar{D}_o^{new} = \bar{D}_o \left( 1 - \frac{\sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i}{\lambda} \right) \quad (12)$$

$\bar{D}_o$  is the current average document size for the overloaded server. In addition in order to estimate the coefficient of variation of service rate it is required to estimate  $\bar{D}_o^{new}$ , which is given by:

$$\bar{D}_o^{new} = \frac{\lambda_o \bar{D}_o^2 - \sum_{j=1}^M \sum_{i=1}^K q_{ij} d_i^2}{\lambda_o} \quad (13)$$

If however we use switching for transferring the load, arrival rate is decreased by the amount of forwarded requests, and new average document size and new average square document size are given by:

$$\bar{D}_o^{new} = \frac{\lambda_o \bar{D}_o - \sum_{j=1}^M \sum_{i=0}^K q_{ij} d_i}{\lambda_o - \sum_{j=1}^M \sum_{i=1}^K q_{ij}} \quad (14)$$

$$\bar{D}_o^{new} = \frac{\lambda_o \bar{D}_o^2 - \sum_{j=1}^M \sum_{i=0}^K q_{ij} d_i^2}{\lambda_o - \sum_{j=1}^M \sum_{i=1}^K q_{ij}} \quad (15)$$

The value of  $\mu_o$  after load sharing can be estimated using Eq. (7). Therefore the following set of new parameters is derived for the overloaded server:

$$\lambda_o^{new} = \begin{cases} \lambda_o & \text{redirection} \\ \lambda_o - \sum_{i=1}^K \sum_{j=1}^M q_{ij} & \text{switching} \end{cases} \quad (16)$$

$$\mu_o^{new} = \begin{cases} \mu_o / \left( 1 - \frac{\sum_{j=1}^M \sum_{i=1}^K q_{ij} d_i}{\lambda_o \bar{D}_o} \right) & \text{redirection} \\ \mu_o \bar{D}_o \frac{\lambda_o - \sum_{j=1}^M \sum_{i=1}^K q_{ij}}{\lambda_o \bar{D}_o - \sum_{j=1}^M \sum_{i=0}^K q_{ij} d_i} & \text{switching} \end{cases} \quad (17)$$

$$\rho_o^{new} = \begin{cases} \rho_o \left( 1 - \frac{\sum_{j=1}^M \sum_{i=1}^K q_{ij} d_i}{\lambda_o \bar{D}_o} \right) & \text{redirection} \\ \rho_o \left( 1 - \frac{\sum_{j=1}^M \sum_{i=1}^K q_{ij} d_i}{\lambda_o \bar{D}_o} \right) & \text{switching} \end{cases} \quad (18)$$

$$(cv_a^{new})^2 = \begin{cases} cv_a^2 & \text{redirection} \\ \frac{cv_a^2 \lambda_o - \sum_{j=1}^M \sum_{i=1}^K cv_i^2 q_{ij}}{\lambda_o - \sum_{j=1}^M \sum_{i=1}^K q_{ij}} & \text{switching} \end{cases} \quad (19)$$

$cv_i$  denotes the coefficient of variation for the inter-arrival time of the requests for document  $i$ . For estimating the coefficient of variation for the service time,  $\bar{D}_o^{new}$  from (14) and (12), and  $\bar{D}_o^{new}$  from (15) and (13) are substituted in (10).

Next is to estimate the new performance parameters for a mirror server  $mirror_j$ . The new value for average document size and average square document size are given by:

$$\bar{D}_j^{new} = \frac{\lambda_j \bar{D}_j + \sum_{i=1}^K q_{ij} d_i}{\lambda_j + \sum_{i=1}^K q_{ij}} \quad (20)$$

$$\bar{D}_j^{new} = \frac{\lambda_o \bar{D}_j^2 + \sum_{i=1}^K q_{ij} d_i^2}{\lambda_o + q_j} \quad (21)$$

Consequently we can estimate the rest of related parameters as following:

$$\lambda_j^{new} = \lambda_j + q_j \quad (22)$$

$$\mu_j^{new} = \mu_j \frac{\lambda_j + q_j}{\lambda_j} \frac{1}{\left(1 + \frac{\sum_{i=1}^K q_{ij} d_i}{\lambda_j \bar{D}_j}\right)} \quad (23)$$

$$\rho_j^{new} = \rho_j \left(1 + \frac{\sum_{i=1}^K q_{ij} d_i}{\lambda_j \bar{D}_j}\right) \quad (24)$$

$$(cv_{a_j}^{new})^2 = \frac{cv_{a_j}^2 \cdot \lambda_j + \sum_{i=1}^K cv_i^2 q_{ij}}{\lambda_j + \sum_{i=1}^K q_{ij}} \quad (25)$$

The coefficient of variation of the service times  $cv_s$  can be derived from (10), where  $\bar{D}^{new}$  and  $\bar{D}^2^{new}$  are given by (20) and (21) respectively.

Finally substituting the new values of  $\rho$ ,  $cv_s$  and  $cv_a$ , in (1) and (2) enable us to estimate new value of response time of the overloaded server  $W_o^{new}$  and new values for response time of each mirror server *mirror<sub>j</sub>* denoted as  $W_j^{new}$ .

In the case of redirection, the actual waiting time perceived by the client is more than what we derive from above equations. From client viewpoint, all the requests arrive at the overloaded server and wait in the queue until their turn to be processed, but after that some of them are processed and others are redirected to the mirror servers and wait in the new queue and processed with different service time. The following equation formulates the waiting time from client viewpoint, after redirecting portion of load to the mirror servers.

$$\begin{aligned} W_o^{client} &= W_{Q_o}^{new} + \frac{\sum_{j=1}^M n_j W_j^{new} + \left(N_o - \sum_{j=1}^M n_j\right) \frac{1}{\mu_o^{new}}}{N_o} \\ &= W_{Q_o}^{new} + \frac{1}{\mu_o^{new}} + \sum_{j=1}^M \frac{q_j}{\lambda_o} W_j^{new} - \sum_{j=1}^M \frac{n_j}{N_o} \frac{1}{\mu_o^{new}} \\ &= W_o^{new} + \sum_{j=1}^M \frac{q_j}{\lambda_o} W_j^{new} - \sum_{j=1}^M \frac{q_j}{\lambda_o \mu_o^{new}} \quad (26) \end{aligned}$$

$W_{Q_o}^{new}$  is the new value of waiting time in the queue. Note that  $n_j = q_j \cdot t$  represents the total number of requests transferred to the mirror server *mirror<sub>j</sub>*, and  $N_o = \lambda_o \cdot t$  the total number of requests arrived at overloaded server within  $t$  seconds.

In this section we estimated the response time of a web server after transferring a portion of its load, and also response time of a mirror server after accepting extra amount of load. The next concern is how much load should be assigned to each mirror server to provide us with best response time, along with minimal charge. The next section deals with this problem.

## 5. Load Sharing

Our motivation for applying partial and temporal mirroring is to transfer minimal load to mirror servers, in order to receive the minimal charge, while keeping the latency of the server under a tolerable threshold. We formulate this problem as an optimization problem with inequality constraints. The constraints are: to keep the latency of the overloaded server and all the cooperating mirror servers under the threshold. A mirror server that gets overloaded is not a choice for improving the performance. The total load transferred to mirror servers is given by:

$$transferred\_load = t \cdot \sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i \quad (27)$$

where  $t$  denotes the period of time the server is overloaded and mirroring continues. This optimization problem is formulated as follows:

$$\min \sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i \quad (28)$$

s.t.

$$W_o^{client} - T_0 \leq 0 \quad (29)$$

$$W_j^{new} - T_0 \leq 0 \quad j = 1, 2, \dots, M \quad (30)$$

$$\left(\sum_{j=1}^M q_{ij}\right) - \hat{q}_i \leq 0 \quad i = 1, 2, \dots, K \quad (31)$$

$$q_{ij} \geq 0 \quad \begin{array}{l} i = 1, 2, \dots, K \\ j = 1, 2, \dots, M \end{array} \quad (32)$$

$\hat{q}_i$  is the total arrival rate for document  $i$ . The task is to determine the value  $q_{ij}$  for  $i = \{1, 2, \dots, K\}$  and  $j = \{1, 2, \dots, M\}$ , which result in the minimal value for the total assigned load.

As we see some of the constraints are nonlinear. The general procedure to deal with this type of problems, is to generate a sequence of convex, explicit subproblems and solve them in an iterative fashion, which is called Sequential Convex Programming (SCP) [15]. Various approximation schemes have been developed for this purpose. We use the globally convergent Method of Moving Asymptotes (MMA) [23] as defined in [24]. To solve the subproblems generated by this algorithm, we have used Lagrangian relaxation problem, solved by conjugate subgradient method. As the result, an optimal answer for  $q_{ij}$ ,  $j = \{1, 2, \dots, M\}$ ,  $i \in \{1, 2, \dots, K\}$  is derived. A copy of the documents is then sent to their selected mirror servers.

To distribute the load among mirror servers, in order to achieve the desired  $q_{ij}$ , we first derive the proportion of load related to each document which is intended to transfer to each mirror server. This is derived from

the following equation.

$$p_{ij} = q_{ij}/\hat{q}_i \quad (33)$$

Any request to the document  $i$ , directed to the overloaded server, is assigned to the mirror server  $mirror_j$ , with the probability  $p_{ij}$ . As the conclusion, the problem of traffic routing which is an issue in mirroring, is reduced to distribute the extra amount of load with the derived probability.

## 6. Experimental Results

The goal of this experiment is to show:

- Mirroring is necessary only if we have a heavy load that causes performance deficiency, to illustrate the feasibility of temporary mirroring.
- Partial mirroring can achieve a reasonable performance. We then compare the cost of mirroring in terms of the transferred load for the case of full mirroring and partial mirroring, and show that partial mirroring reduces the cost, while assuring to maintain the performance.

Note that full mirroring refers to conventional mirroring, where the load is equally distributed among mirror servers. Consequently probability of assigning a request corresponding to document  $i$  to server  $mirror_j$  is  $\frac{1}{M+1}$ . Partial mirroring in the other hand refers to our approach, where probability of assigning each request to each mirror server is derived from (33). Interestingly as we see later, partial mirroring guarantees to maintain the performance under heavy load even if full mirroring fails to maintain the performance for the same load. In the following we first describe the test environment and the measurements.

### 6.1 Test Environment

In our experiment, we use a group of three servers, running Apache software (version 1.3.19), each running on a Solaris 2.8 operating system. One of the servers is intended to simulate the behavior of web servers and the other two are mirror servers. We use a traffic generator benchmark, which will be described in detail later, to generate a heavy load on the server. Since the experiment is applied in a LAN, we use *dummynet* [21], as the network emulator, to simulate the characteristic of a WAN. Figure 5 shows this simulation. We define a multipath channel between the benchmark and the web server to account for a variety of possible paths. In particular we create four pipes, each representing a class of traffic on a WAN. The bandwidth for these pipes are: 128 Kb, 256 Kb, 625 Kb and 1 Mb. Delay in terms of msec defined by RTT (Round Trip Time) varies from 75 to 200 to account for normal delays and long delays. Note that as we see in Fig. 5 the requests sent to the

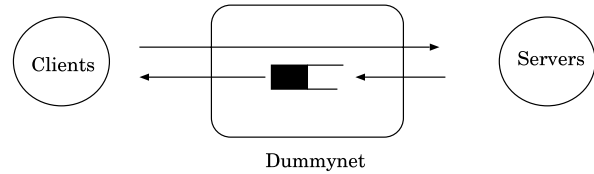


Fig. 5 Test environment.

Table 1 Workload generated by the benchmark.

| File size (KB) | Probability of access |
|----------------|-----------------------|
| 0.5            | 0.35                  |
| 5              | 0.5                   |
| 50             | 0.14                  |
| 500            | 0.009                 |
| 5000           | 0.001                 |

server are not queued. This is because we tune the request rate in our benchmark to generate a simulated inter-arrival rate on the server. However, when serving the requests, packets are queued before getting to client to simulate the WAN bandwidth and latency.

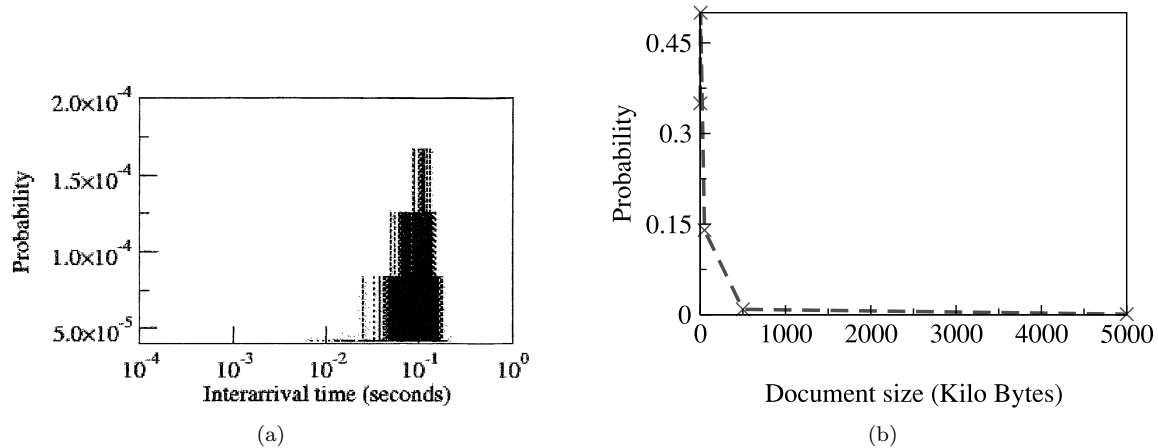
### 6.2 Traffic Generator

Our traffic generator benchmark written in C runs on FreeBSD operating system. The workload generated by this benchmark is intended to simulate the real workload of web servers. It is shown [11] that file size distribution of the WWW has a heavy tailed distribution: most documents are small (a few kilo bytes) but he number which are very long tend to contribute the majority of traffic. The file set that simulates this characteristics for our benchmark is adopted from webstone benchmark [26], and the setting is given in Table 1. Also the probability of retrieval of these files is illustrated in Fig. 6(b).

Inter-arrival time of WWW requests can be modeled by a Weibull distribution, which is a rather flexible distribution that captures small to large inter-arrival times [14]. Applications such as Web, create inter-arrival times from very short to very long. Very short inter-arrival times are mainly due to the embedded objects within a homepage (e.g., images). Currently these embedded objects are retrieved via parallel TCP connections, limited to four connections. Use of persistent connections [20], reduces this number to two, but still leaves the possibility of small inter-arrivals. Moreover, Weibull distribution has the advantage to include exponential distribution, which traditionally has been used to model the arrivals, as a special case. Mathematically the distribution is defined as:

$$f(x) = \frac{\beta x^{\beta-1}}{\eta^\beta} e^{-\left(\frac{x}{\eta}\right)^\beta} \quad \beta, \eta > 0, x \geq 0 \quad (34)$$

where  $\beta$  is shape parameter defining the shape of the distribution and  $\eta$  is scale parameter.  $\beta > 3.06$  creates a left heavy tailed distribution, which matches the



**Fig. 6** Benchmark characteristics: (a) probability density function for inter-arrival time for ( $\lambda = 10$ ) req/sec (b) probability density function for document size.

characteristics of inter-arrivals [14]. The mean of this distribution is given by:

$$E(x) = \eta\Gamma\left(1 + \frac{1}{\beta}\right) \quad (35)$$

In our benchmark  $\beta = 5$  simulates the left heavy tail nature of inter-arrivals, and  $\eta$  is assigned a value to simulate the intended request rate ( $\lambda = 1/E(a)$ ). In fact in our experiment,  $\eta$  is altered, to gradually increase the request rate.

The density function of the probability of inter-arrival time and document size generated by this benchmark is shown in Figs. 6(a), (b).

This benchmark generates simultaneous clients each running as separate threads. The waiting time is measured, as perceived by the client. Each client thread measures the waiting time from when it sends the request until when it receives the last byte. In the case of redirection, it measures the time from when it sends the request to the original server until when it receives the last byte of the requested document from the mirror server. The benchmark reports the average waiting time, as perceived by the clients.

### 6.3 Measurements

In order to measure the inter-arrival statistics, we implemented a packet capture program to collect all the incoming packets to the port 80 (HTTP port) which have their SYN flag set. These packets represent HTTP incoming requests. To measure the mean of inter-arrival time, we measure the time between each two consequent requests, and update the mean and variance of the inter-arrival times sequentially.

Measuring service time is a little more complicated. The response time of a web server refers to the time that takes for the HTTP process to send the whole document to the client. Therefore we measure the time from when the HTTP process starts processing request,

until when the last FIN is acknowledged, indicating the end of data transfer. We modified Apache to assign a time stamp to each connection as soon as it accepts a new request. To find out when a connection is closed we capture all the incoming and outgoing packets from port 80, that have their FIN flag set. As soon as the last FIN on a connection is acknowledged the connection is considered closed and the time between accepting the request and closing the connection is reported as service time.

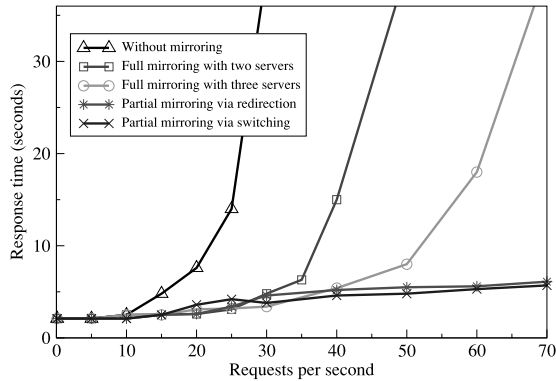
For the statistic related to documents, we added a code to Apache to update these statistics whenever a document is retrieved. The maximum number of connections in our Apache server is set to 50 connections. This defines the number of servers  $c$  in estimating waiting time.

### 6.4 Results

We use our web traffic generator benchmark and gradually increase the load of the web server. Changes in the traffic are applied every 30 minutes. Our earlier experiments in shorter duration (5 minutes), although confirms the current results, but is insufficient as the the proof of effectiveness of our approach, since the load does not sustain long enough to project the characteristic of the real traffic.

We define the threshold of 5 seconds as an upper limit of tolerable latency. Each experiment is repeated for the case that load is redirected to the mirror servers, and the case that load transfer is applied via switching. We have simulated the switching in the client side (in the benchmark), and our goal is to show if requests are transferred to the mirror server, via a switch, before getting to the overloaded server, performance can be improved.

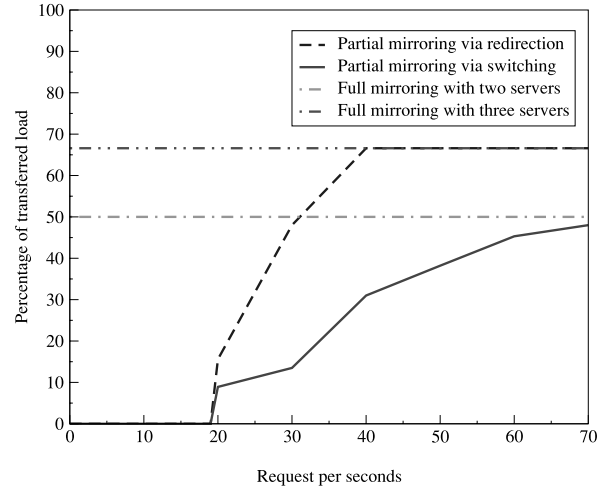
Figure 7 compares the latency with and without using partial and temporal mirroring, when load is redirected to the mirror servers, and when switching has



**Fig. 7** Comparing latency with and without applying partial mirroring.

been used to transfer the load. Waiting time is what is reported by the benchmark and represents what actually the client observes. Note that in case of full mirroring the probability of assigning a request to mirror servers is simply  $M \cdot \frac{1}{M+1}$ , where  $M = 2$  and therefore probability of assigning a request to mirror servers is  $\frac{2}{3}$ . In the other hand, this probability for case of partial mirroring in our approach, is derived from (33) and is shown in Figs. 9(a) and (b). As we see in Fig. 7, to provide an 5 seconds latency, mirroring is required only when the number of incoming requests per second exceed 20 (around 2 million hits per day). This request rate corresponds to the peak time. In fact we need to use mirroring services only in busy hours. Also, as illustrated in this figure, partial mirroring with two mirror servers allows maintaining a 5 seconds latency for over 5.6 million hits per day (65 requests per second), while applying full mirroring with one mirror server can maintain this latency only for up to 35 requests per second. Full mirroring with two mirror servers increase this number to almost 50 requests per second, which is still less than what can be achieved by partial mirroring. The reason is, in full mirroring we simply distribute the load and the request rate equally among the mirror servers, consequently having three servers that receive 50 requests per second for the average document size generated by the benchmark, is almost similar to having one web server that receives 20 requests per second<sup>†</sup>, which according to Fig. 7 fails to provide the 5 second latency. However our optimization algorithm generates different average document sizes on mirror servers and allows us to distribute the request rate and the load, so that we have minimum transferred load while maintaining 5 seconds latency.

Figure 8 compares the cost in terms of transferred load for full mirroring and partial mirroring. As illustrated in this figure, although using redirection as the load transfer method, can alleviate the cost in contrast to full mirroring, using a switch is recommended since it can effectively reduce the cost. As we see in this figure, for up to 50 requests per second in contrast to



**Fig. 8** Comparing the amount of transferred load to the mirror servers for the case of full mirroring, partial mirroring via redirection, and partial mirroring via switching.

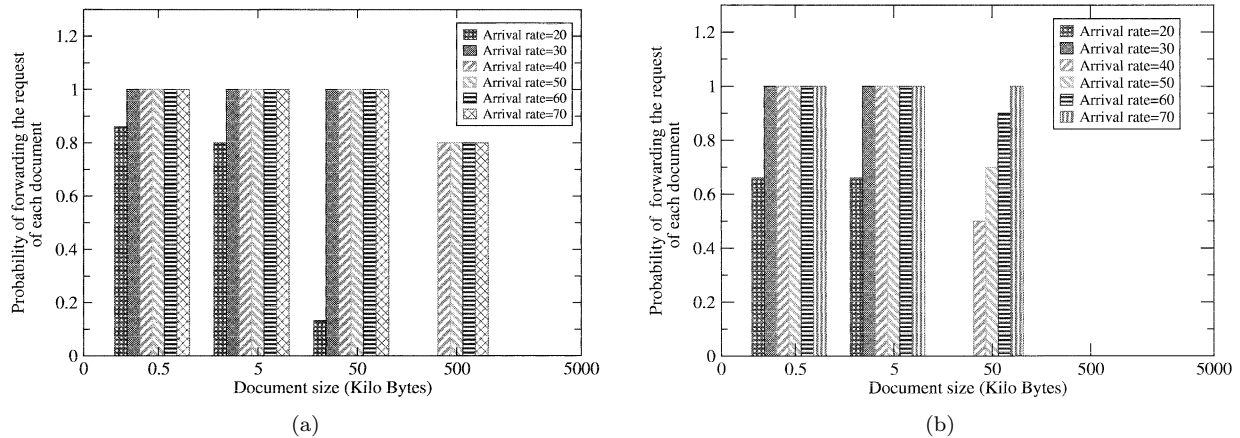
full mirroring with 3 servers, using a switch can reduce the transferred load to half (and if the request rate is less than 40 it is reduced to one third), indicating the fact that mirror servers can be shared among two (to three) overloaded servers in the worst case (i.e., all the servers are overloaded by up to 50 requests per second). For more than 50 requests per second, partial mirroring is the only choice, full mirroring fails to maintain the performance. The main conclusions are as follows:

- Partial mirroring can maintain the desired latency, even under the heavy traffic when full mirroring fails to maintain this latency.
- According to our experiment with set of two mirror servers, partial mirroring can reduce the cost to half (or one third if the arrival rate is less than 40 requests per second). Allocating more mirror servers is expected to give better results.
- This approach allows applying mirroring only during peak time, therefore when the server is lightly loaded there is no need to apply mirroring and mirroring services is given to other overloaded servers.

Finally to derive an insight about how the load is transferred, Fig. 9 illustrates the probability of assigning a request of each document to mirror servers, as derived from (33) in case of redirection, and Fig. 9(b) illustrates this probability for the case of switching.

As can be concluded from Fig. 9(a) and Fig. 9(b) in the case of redirection, where the only way to reduce  $\rho$  and therefore latency is to tune the service rate  $\mu$ , it is preferred to transfer the traffic generated by bigger files in contrast to switching where it is preferred to transfer the load generated by the smaller files.

<sup>†</sup>We expect to handle  $20 \cdot 3 = 60$  request with three servers, however the overhead of request forwarding decreases this amount to 50.



**Fig. 9** Probability of forwarding the request corresponding to each document to the mirror servers (i.e.,  $\sum_{j=1}^M p_{ij}$ ). (a) via redirection (b) via switching.

## 7. Conclusions

This article introduced a new approach based on sharing mirror servers, to reduce the cost of mirroring. We showed that by transferring load to mirror servers we can tune the latency of web servers. We derived the minimal amount of load required to be transferred to mirror servers to keep the latency within a tolerable threshold. The dynamic mirror server allocation, allows us to take traffic situation into consideration, when choosing the mirroring partners. Moreover, dynamic load assignment, enables us to decide about the documents to be mirrored, based on the load generated by each document at any time. Consequently at any time, we can assign the optimal load to the optimal mirroring partners. This has two main implications: (1) the number of devoted mirror servers can be minimized using this approach, (2) it provides a means of sharing mirror servers in order to decrease the cost. As the result of this work, we will have a balanced load among the servers, and a smoother traffic along the Internet, as well. Our experiment, holds promise of keeping latency under a desired threshold, while maintaining minimal cost of mirroring.

## References

- [1] E. Anderson, D. Patterson, and E. Brewer, "The magi-router, an application of fast packet interposing," <http://HTTP.cs.Berkeley.EDU/~eanders/projects/magicrouter/osdi96-mr-submissions.ps>
- [2] A. Bestavros, "WWW traffic reduction and load balancing through server-based caching," *IEEE Concurrency*, vol.5, no.1, pp.56–67, Jan.–March 1997.
- [3] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science*, Chapter 6, pp.233–237, Wiley-Interscience Publication, 1998.
- [4] T. Brisco, DNS Support for Load Balancing, RFC-1794, <http://www.cis.ohio-state.edu/htbin/rfc/rfc1794.html>, April 1995.
- [5] R.B. Bunt and D.L. Eager, "Achieving Load Balance and Effective Caching in Clustered Web Servers," *The 4th International Web Caching Workshop*, pp.159–169, San Diego, California, April 1999.
- [6] ClarkNet traffic, <http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html>
- [7] R. Caceres, F. Douglis, and A. Feldmann, "Web proxy caching: The Devil is in the details," *Performance Evaluation Review*, vol.26, no.3, pp.11–15, Dec. 1998.
- [8] V. Cardellini, M. Colajanni, and P.S. Yu, "Dynamic load balancing on web-server systems," *IEEE Internet Computing*, vol.3, no.3, pp.28–39, May/June 1999.
- [9] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," *Proc. 2000 IEEE Infocom Conference*, pp.1742–1751, Tel Aviv, Israel, March 2000.
- [10] M.E. Crovella and R. Carter, "Dynamic server selection in the Internet," *Proc. IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, pp.158–162, Aug. 1995.
- [11] M.E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes," *IEEE/ACM Trans. Networking*, vol.5, no.6, pp.835–846, Dec. 1997.
- [12] O.P. Damani, P.E. Chung, Y. Huang, C. Kintala, and Y. Wang, "One-IP: Techniques for hosting a service on a cluster of machines," *Sixth International World Wide Web Conference*, pp.1019–1027, Santa Carla, 1997.
- [13] J. Dille, "Web server workload characterization," *HP Labs Technical Reports*, HPL-96-160, Dec. 1996.
- [14] A. Fledmann, *Characteristics of TCP Connection Arrivals*, Technical Report, AT&T Labs Research, Dec. 1998.
- [15] C. Fleury, "Sequential convex programming for structural optimization problems," in *Optimization of Large Structural Systems*, volume I of NATO ASI, ed., G. Rozvany, pp.531–553. Kluwer Academic Publishers, 1993.
- [16] S. Floyd, "Connections with multiple congested gateways in packet switching networks, Part1: One-way traffic," *Computer Communications Review*, vol.21, no.5, pp.30–47, Oct. 1991.
- [17] H. Guyennet, F. Spies, and M. Trehel, "Modeling and simulation of dynamic load balancing using queuing theory," *Parallel Algorithms and Applications*, Gordon and Breach science publishers, vol.5, nos.1–2, pp.199–218, 1995.
- [18] B. Liu, "Web traffic latency: Characteristics and impli-

cations," *J. Universal Computer Science*, vol.4, Issue 9, pp.763–778, 1998.

- [19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *SIGCOM'98*, pp.303–314, Sept. 1998.
- [20] V.N. Padmanabhan and J. Mogul, "Improving HTTP latency," *Computer Networks and ISDN Systems*, vol.28, no.1/2, pp.25–35, Dec. 1995.
- [21] L. Rizzo, "Dumynet: A simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol.27, no.1, pp.31–41, Jan. 1997.
- [22] J.W. Roberts, "Traffic theory and the internet," *IEEE Commun. Mag.*, vol.39, no.1, pp.94–99, Jan. 2001.
- [23] K. Svanberg, "Method of moving asymptotes—A new method for structural optimization," *International J. Numerical Methods in Engineering*, vol.24, pp.359–373, 1987.
- [24] K. Svanberg, "The MMA for modeling and solving optimization problems," *Proc. 3rd WCSMO (The World Congress of Structural and Multidisciplinary Optimization)*, 4EOM3, Buffalo, New York, May 1999.
- [25] M. Trehel, C. Balayer, and A. Alloui, "Modeling load balancing inside groups using queuing theory," *PDCS'97*, 10th International Conference on Parallel and Distributed Computing System, New Orleans, Louisiana, Oct. 1997.
- [26] Webstone, <http://www.mindcraft.com/webstone>
- [27] D. Wessels, *Intelligent Caching for World-Wide Web Objects*, Master Thesis, University of Colorado, Interdisciplinary Program in Telecommunication, 1995.
- [28] D. Wessels and K. Claffy, "Evolution of the NLANR cache hierarchy: Global configuration challenges," <http://www.nlanr.net/Papers/Cache96>, 1996.
- [29] W. Whitt, "Towards better multi-class parametric-decomposition approximations for open queuing networks," *Annals of Operations Research*, vol.48, pp.221–248, 1994.
- [30] K.J. Worrell, *Invalidation in large scale network object caches*, Master thesis, University of Colorado, 1994.



**Behrouz Homayoun Far** received BSc. and MSc. degrees in Electronic Engineering in 1983 and 1986, respectively, from Tehran University, Iran. He has received his Ph.D. degree from Chiba University, Japan, in 1990. He is currently an Associate Professor at the Department of Electrical and Computer Engineering, University of Calgary, Canada, where he is the coordinator of the Intelligent Systems Group at The University of Calgary.

The research fields of his interest are automatic programming, software quality management and distributed AI. Dr. Far is a member of the ACM, IEEE Computer society, JSAI, and IPSJ.



**Jingde Cheng** received the B.E. degree in computer science from Tsinghua University in China in 1982, and the M.E. and D.E. degrees in computer science from Kyushu University in Japan in 1986 and 1989, respectively. He is currently a professor of computer science at Graduate School of Science and Engineering, Saitama University in Japan. Before he joined Saitama University in 1999, he was a research associate (1989–1991), an

associate professor (1991–1996), and a professor (1996–2000) of Kyushu University. His current research interests include ampliative reasoning and relevant reasoning, relevant logic and its applications, epistemic programming for scientific discovery, autonomous evolution of knowledge-based systems, design, development, maintenance, and evolution methodologies for highly reliable and secure software systems, and information security engineering. He is a member of IEEE-CS, ACM, AAAI, IPSJ, JSSST, JSAI, and JSSM.



**Shadan Saniepour** received BSc. degree in Electronic and Telecommunication Engineering in 1993 from Iran University of Science and Technology, Iran. She has received her MSc. and Ph.D. degrees from Saitama University, Japan in 1999 and 2002, respectively. She is currently working at the network-engineering department of the JENS corporation-Japan. Her research interests are in network and web server traffic engineering and performance management.

performance management.