Analysis of Task Assignment Policies in Scalable Distributed Web-server Systems*

Michele Colajanni
Dip. di Informatica, Sistemi e Produzione
Università di Roma – Tor Vergata
colajanni@uniroma2.it

Philip S. Yu and Daniel M. Dias
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
psyu@us.ibm.com

Abstract

A distributed multi-server Web site can provide the scalability necessary to keep up with growing client demand at popular sites. Load balancing of these distributed Web-server systems, consisting of multiple Web servers for document retrieval and a Domain name server (DNS) for address resolution, opens interesting new problems. In this paper, we investigate the effects of using a more active DNS which, as an atypical centralized scheduler, applies some scheduling strategy in routing the requests to the most suitable Web server.

Unlike traditional parallel/distributed systems in which a centralized scheduler has full control of the system, the DNS controls only a very small fraction of the requests reaching the multi-server Web site. This peculiarity, especially in the presence of highly skewed load, makes it very difficult to achieve acceptable load balancing and avoid overloading some Web server.

This paper adapts traditional scheduling algorithms to the DNS, proposes new policies, and examines their impact under different scenarios. Extensive simulation results show the advantage of strategies that make scheduling decisions on the basis of the domain that originates the client requests, and limited server state information (e.g. whether a server is overloaded or not). An initially unexpected result is that using detailed server information, especially based on history, does not seem useful in predicting the future load, and can often lead to degraded performance.

Index Terms: *Distributed systems, Load balancing, Performance analysis, Scheduling algorithms, Web servers.*

*©1998 IEEE. Published in *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 6, June 1998. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 732-562-3966.

1 Introduction

The rapid expansion of World Wide Web has led to exponential growth in the request rate to some popular sites. The combination of increased usage of network bandwidth and overloaded servers may cause users to spend much of their session time waiting for access to documents. For these reasons, the replication of information across *independent* or *coordinated* mirrored servers is becoming a common choice for most popular sites [3, 21]. Various reasons indicate that a coordinated architecture, namely a *distributed Web-server system*, is the most promising solution. A distributed multi-server system is more scalable and fault-tolerant than any centralized counterpart. Furthermore, it maintains a single user-view interface that is, a single URL, and its load can be balanced. In contrast, the independent mirrored-server architecture provides a list of independent URL sites that are manually selected by the users. This solution does not provide the transparency for users and precludes any control of the request (re)distribution by the Web-server system. However, if the components of the distributed Web-server system are not well designed and the results are not evaluated carefully, the complexity and irregularities of World Wide Web could degrade the performance of a coordinated architecture.

Existing installations of distributed multi-server Web sites are the NCSA HTTP Server [16] and the IBM Web server built on an SP-2 machine [13, 17]. A commonly used approach for load balancing is the Round-Robin Domain Name Server (RR-DNS) technique described in [16, 1]. In [13] an alternative, referred to as the TCP router approach, is considered which uses a TCP front-end to distribute incoming requests among a set of clustered servers. Since the DNS technique can be applied to both locally and geographically distributed Web-server systems, in this paper, we propose and evaluate policies that can improve the load balance through an extension of the DNS technique.

The distributed Web-server system, illustrated in Figure 1, consists of multiple, replicated Web servers (WS) that furnish the `html` documents, and a cluster DNS that translates the logical `http` address (*name*) into the IP-address of one of the Web servers of the logical cluster. The DNS can use various scheduling policies to map different clients to the different servers in the cluster. However, the Web browsers at the clients and several name servers on the path from clients to the DNS typically cache the name-to-address mapping returned by the DNS. When an address resolution is found in one cache of the path, the page request is directly sent to the address of the indicated WS, bypassing the address resolution in the DNS. Otherwise, the name request has to be resolved by the cluster DNS. As a result of address caching, a large number of clients in the domain behind the same name server are mapped to the same WS, leading to a load imbalance among the servers, as quantified in [13]. To reduce load imbalance, while returning the physical address of a WS, the DNS could apply some *scheduling* policy that routes the page requests to the "best" WS.

Unlike traditional parallel/distributed systems in which a centralized scheduler has full control on the incoming requests [14, 7, 20, 15], due to caching in the name servers and at the Web browsers, the DNS typically controls a small fraction, often on the order of a few percent, of the requests reaching the Web site. The non-uniformity of the load introduces additional degrees of complexity to the request scheduling issue. These peculiarities create an interesting challenge to any scheduling

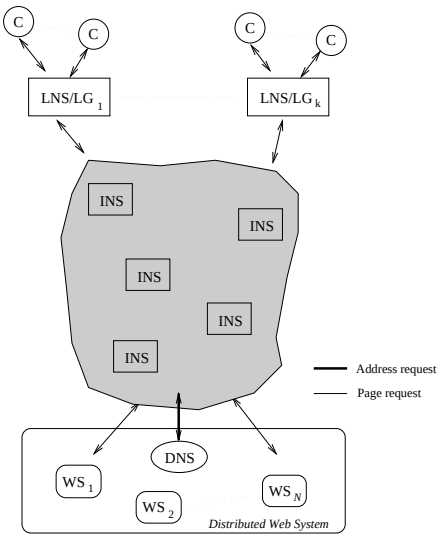

Figure 1: System model and network flow.

algorithm that the DNS may employ. This paper analyzes traditional algorithms, proposes new ones, and compares their performance under different scenarios. The basic premise of our work is the compatibility with existing Web standards so that the proposed DNS scheduling policies could be easily adopted.

The rest of the paper is organized as follows. In Section 2, we present a model for distributed Web-server systems that is suitable to the performance analysis of DNS scheduling policies. In Section 3, we describe algorithms that can be used by the DNS to assign the requests to the servers. In Section 4, we examine the metrics and the model parameters that we used in the experiments. In Section 5, we discuss the experimental results which are achieved for different systems and scenarios. In Section 6, we outline our conclusions.

2 Model Description

The model reflects the focus of the paper on a multi-server Web site and the DNS scheduling algorithm. Since the analysis is carried out from the point of view of a particular Web-server, we detail the aspects that affect that multi-node Web-server system without attempting to model the entire Internet with thousands of other Web-server systems. As illustrated in Figure 1, the system we are considering is composed of a set of *clients* that are connected to Internet through *local name server(s)* (LNS) and *local gateways* (LG). From the point of view of the Web-server system, the client is only identifiable from its domain. The distributed Web-server system consists of several Web servers (sharing the same URL) and a cluster DNS that receives all initial address resolution requests coming through LGs. Several *intermediate name servers* (INS) typically exist between a LG and a WS. While returning the address of a WS, the DNS specifies a *time-to-live* (TTL) that is, the duration of time for which a name-to-address mapping is cached at LNSs and INSs.

In addition to the normal task of resolving the URL address into the IP-address of a WS, the DNS of a distributed Web-server system can play the role of a centralized scheduler. In the selection of the address of a WS, the DNS could implement some policy that attempts to balance the load among multiple servers or avoid some WS from becoming overloaded. However, to achieve these goals, the DNS-scheduler faces the following obstacles which make it different from a normal scheduler.

- Address caching limits the control of the DNS on the requests reaching the Web site. Therefore, most factors that affect the system performance are independent of the DNS decisions.
- During the TTL period, subsequent Web requests from a LG arrive to the same WS. This can cause high skews especially if the LG serves many clients.
- Using TTL$\approx$ 0 could cause the DNS to become a bottleneck. In addition, most name servers use a minimum TTL value if the received TTL is considered too small ("we cannot make people ask us for the address, and we cannot control what they do with the address" [18]).

This TTL period for caching name-to-address mapping differentiates the DNS scheduling problem from the conventional scheduling problem in distributed systems which can be viewed as a special case with TTL equal to zero. (This is approximately but not strictly correct because browsers cache the mapped address, and subsequent requests from the same browser would be sent to the same node.) In the conventional distributed system, the queue length at each server can usually provide a good indication of the server load. In the distributed Web-server system, the queue length at each server does not reflect how many *committed* future arrivals will occur due to the TTL effect on past assignments. Consider a server with low utilization. Suppose that the DNS made a burst of name-to-address mappings to that server; the server's utilization may still appears to be low for a short while, and could then become overloaded after INSs and LNSs continue to map requests to this server based on their address caching for the TTL period.

Many existing distributed Web sites assign the requests arriving at the DNS in a round-robin manner among the WSs [16, 1]. In this paper, we demonstrate that the Round-Robin Domain Name Server policy (RR), referred to as DNS-RR, works well only if the clients behind each LG are uniformly distributed. However, for the more realistic hypothesis of a non-uniform distribution (e.g., geometric, *Zipf's*), round-robin assignments of the clients to Web servers can perform poorly.

2.1 Distributed Web-server system

The distributed Web-server system under consideration is composed of N Web servers and a cluster DNS that receives all initial address resolution requests coming through LGs. Since the servers are assumed to be mirrored, the page requests can be served by any WS.

2.2 Network model

Since the focus of this paper is on the performance of the Web-server system, we do not evaluate the user latency time over the network. We model only the Internet components that impact the

performance of the distributed Web-server system such as the *intermediate name servers*. The INSs could handle some address resolution request such that the DNS is bypassed. The consequence is that the DNS looses the control on subsequent Web requests coming after an address resolution. When the DNS returns the IP-address of one of the WS in the system, it also provides a TTL to the name servers along the path to the client so that intermediate and local name servers can cache the name-to-address mapping for the TTL period chosen by the DNS (or a minimum TTL they impose, if the DNS value is too low).

Details of the messaging traffic are not modeled for the following reasons. As we will see, the scheduling policies that require more messages among Web servers and DNS already show worse performance, while the better policies incur only a communication load that is negligible if compared to client requests for Web documents. Therefore, capturing the communication costs would not affect our conclusions. Furthermore, accurate models of Internet traffic are still under study [8, 5, 10]. Consequently, in this paper we do not consider the class of scheduling algorithms that take into account geographical locality. Future research aims at relaxing this assumption.

2.3 Web clients

The *clients* belong to different domains identified in Figure 1 by the LNS/LG. Typically, network service providers and corporations have a set of LNSs and are connected to the network through LG, such as firewalls or socks/proxy servers. Under existing Web standards, the address of this domain is the only client information which is accessible to the DNS when it receives an address resolution request. Therefore, for the purposes of our analysis, a LG is representative of the client *domain*, and throughout the paper we will use both definitions equivalently. We also consider singleton LGs which represent stand-alone clients. Client performance issues are indirectly addressed, because minimizing server overloading improves the throughput of the overall system and reduces the client's mean latency time and unsuccessful responses.

An important issue is how the clients are distributed among the domains. A detailed analysis for popular Web sites is not yet available in literature. However, various studies indicate that if one ranks the popularity of domains by the frequency of their accesses to the Web server, the client distribution is a function with a small number of large values (corresponding to popular Internet service providers, large research institutes and corporations behind firewalls), and a long tail assuming small values. For example, a workload analysis on academic and commercial Web sites shows that on average 75% of the client requests come from only 10% of the domains [2]. A good distribution that fits the non-uniformity of this function is the Zipf-like distribution that is, a parametric distribution where the probability of selecting the i-th item is proportional to $1/i^{(1-x)}$. This distribution for a set of L items is given by $q_i = c/i^{(1-x)}$ for each $i \in \{1, \ldots, L\}$, where $c = 1/\left[\sum_{i=1}^{L} 1/i^{(1-x)}\right]$ is a normalization constant [26]. The *pure* Zipf's function (that is, with parameter $x = 0$) is commonly adopted in various social contexts to model the distribution of people choices, such as the number of requests for a Web page [11]. Figure 2 shows how clients are partitioned among $L = 20$ subdomains in case of a geometric distribution with parameter $p = 0.3$, a pure Zipf's distribution, a Zipf with $x = 0.5$, and a uniform distribution (that is, a Zipf with

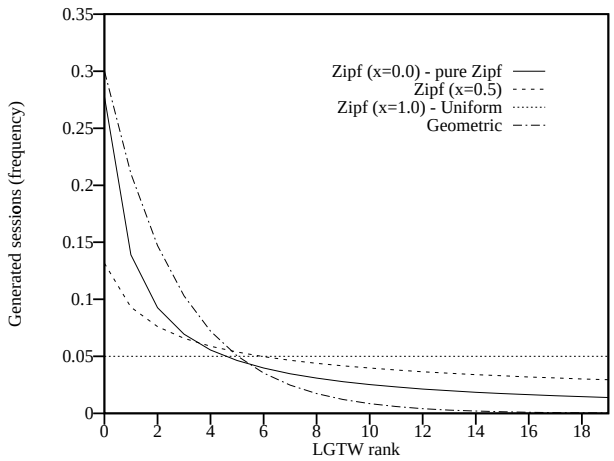

Figure 2: Various distributions of clients among 20 domains.

parameter $x = 1$). Note the long tail of the Zipf distributions that led us to prefer this function for modeling the partition of clients among domains for the base case of the performance study. In Section 5.4, sensitivity analysis is provided on the number of LGs and various distributions of clients among the LGs.

2.4 Load model

We model each client *session* at the same Web site as characterized by one address resolution and several Web page *requests*. In the first phase, the client obtains the address of one WS of the cluster through the DNS address resolution process. Subsequently, the client submits multiple Web page requests that are separated by a mean inter-request time (*think time*). No caching of documents is considered. The client first gets the main `html` page, then transparently issues one `http` request for each object contained in a Web page. We will refer to these separate accesses to the Web server as *hits*.

This model is representative of present Web load, where `html` and image documents are more than 90% of the total requests to the Web servers [2]. Multimedia objects like sound and video account for only 1–2% of the requests, even if these files may reach a larger percentage of the bytes transferred. The growth in the use of multimedia objects and Java applets will surely increase the relative percentage of these requests. By properly selecting the workload parameters to model the number of requests per session, the number of hits and the service time, different types of workloads that a client session may impose on a WS can be represented. A detailed model of the dimension and nature of Web files is orthogonal and beyond the scope of this paper. A sensitivity analysis on the number of requests per session, the number of hits per request and its service time confirmed the main conclusions of this paper.

3 Scheduling algorithms

In this section we present some strategies that aim at improving the performance of distributed multi-server Web sites that are likely to perform poorly under the naive RR-DNS. All the proposed scheduling algorithms tend to use additional state information in the mapping of URL names to IP-addresses of Web servers. The system state information accessible to the DNS is of two kinds.

Information on LG. The existing protocol allows the DNS to identify the origin of each request asking for an address resolution. Moreover, the DNS can collect information on LGs from WSs for various statistics.

Information on WS load. A variety of data can be obtained from each WS. This can range from a simple *asynchronous alarm* from a WS to signal the DNS that the WS is becoming heavily loaded, to *frequent messages* from WS to keep DNS informed on detailed processor loads and history of server state.

Depending on the type of information used, we can partition the DNS scheduling policies into three main classes: only LG information, only WS information, combination of LG and WS information. The amount of information necessary for each algorithm will be used to further distinguish the policies into sub-classes. Determining the usefulness of different levels of information is one of the main purposes of this paper. We note that, although the proposed algorithms vary in terms of implementation complexity and communications load, all of them are applicable to a real-time environment and do not require changes to existing Web standards. The issue of how to accurately estimate the necessary state information is analyzed in [6].

3.1 Algorithms using domain information

The first class of algorithms estimates the average number of subsequent client requests from each LG to a WS resulting from a name-to-address mapping. The DNS tracks the number of name-to-address mapping requests it received from each LG during each measurement period. Let N_j^M be the number of name-to-address mapping requests that the DNS receives from LG_j. Each WS tracks the number of requests it received from each LG during the corresponding period. Let $N_{i,j}^R$ be the number of Web requests that WS_i received from LG_j. Periodically (i.e. for each measurement period), the DNS collects the $N_{i,j}^R$ information from each WS. DNS uses $\sum_i N_{i,j}^R / N_j^M$ to estimate the average number of Web requests (η_j) from LG_j per mapping. (Not all Web requests will fall into the measurement period but some of the requests from the previous mappings may continue to fall into this interval. So statistically it should average out.) We refer to the value η_j as the *hidden load weight* of LG_j since it is transparent to the DNS. A WS is selected on the basis of the address of the LG that has originated the client request and its hidden load weight. No tracking of load level of the Web servers is required.

Two-tier-Round-Robin (RR2). This algorithm is motivated from the observation that the hidden load weight of each LG can be very different. For this reason, we assume a simple partition

of the domains into two classes: *normal* LGs, and *hot* LGs (such as popular providers and large companies and institutes). Based on the origin domain of the requests, the RR2 strategy applies a round-robin policy to each class of LG separately. That is to say, under the RR2 strategy, a separate round-robin scheduling list is maintained for the requests from each class. (The round-robin of each class can start from a different WS.) The objective is to reduce the probability that the hot domains are assigned too frequently to the same WS.

For example, assume two WSs and four LGs, where LG_1 and LG_3 are hot LG and LG_2 and LG_4 are normal LGs. We denote the j-th name-to-address mapping request from LG_i to be $R^A_{i,j}$. Assume a sequence of name-to-address mapping requests to the DNS (in order of arrivals) is $R^A_{1,1}, R^A_{2,1}, R^A_{3,1}, R^A_{4,1}, R^A_{1,2}, R^A_{2,2}, R^A_{1,3}$, $R^A_{1,4}$, and $R^A_{3,2}$. Under RR2, there are two independent round-robin scheduling lists: one (hot) list for requests from LG_1 and LG_3 and another normal list for LG_2 and LG_4. The requests on the hot list include $R^A_{1,1}, R^A_{3,1}, R^A_{1,2}, R^A_{1,3}, R^A_{1,4}$ and $R^A_{3,2}$. The requests on the normal list include $R^A_{2,1}, R^A_{4,1}$, and $R^A_{2,2}$. Assume under RR2, the round-robin scheduling of the hot list starts at WS_1 and the other list starts at WS_2. RR2 assigns to WS_1: $R^A_{1,1}, R^A_{1,2}$, and $R^A_{1,4}$ from the hot list and $R^A_{4,1}$ from the normal list. RR2 assigns to WS_2: $R^A_{3,1}, R^A_{1,3}$, and $R^A_{3,2}$ from the hot list and $R^A_{2,1}$ and $R^A_{2,2}$ from the normal list. The assignment policy divides the request load from the hot LGs (similarly for the normal LGs) fairly evenly among the two WSs. Under the RR strategy, the assignment is as follows. $R^A_{1,1}, R^A_{3,1}, R^A_{1,2}, R^A_{1,3}$, and $R^A_{3,2}$ from the hot list are assigned to WS_1. There is no request from the other list assigned to WS_1. On the other hand, only one request ($R^A_{1,4}$) from the hot list and all requests from the normal list ($R^A_{2,1}, R^A_{4,1}$, and $R^A_{2,2}$) get assigned to WS_2. The load can thus become very unbalanced among the two WSs.

For default, we set the *class threshold* T_C to $(1/\#LG)$, where $\#LG$ is the average number of domains connected to the Web-server system. The domain LG_k belongs to the hot class, if the *relative* hidden load weight $\eta_k / \sum_h \eta_h$ is larger than T_C. For example, if $\#LG = 20$, all domains that generate more than 5% of the requests belong to the hot class.

Note that several variants of this algorithm can be derived in practice. As an example, some LGs can always be considered as hot, while others may join the hot class only occasionally. Moreover, the threshold T_C can be set differently, and/or more than one threshold can be established, that is, *multi-tier-round-robin*. However, we will focus on the RR2 because we have observed in a joint study that it is not convenient to partition the domains into more than two classes.

Dynamically Accumulated Load (DAL). As in RR2, DAL needs to estimate the hidden load weight of each LG. While the RR2 policy uses the hidden load weight estimation to do a binary partitioning of the domains, DAL uses the hidden load weight to estimate future loads (due to previous DNS assignments) in making scheduling decisions. For this reason, the accuracy of the estimates becomes more critical than for RR2, and additional information from the WS could be useful as discussed in [6]. The DNS accumulates the hidden load weights of the previous assignments in a *bin* for each WS. At each request of new address

resolution, the DNS chooses the WS with the lowest bin value. Furthermore, each time the DNS makes a WS selection, it increases the bin value of the chosen WS by the hidden load weight of the requesting LG to reflect the number of subsequent requests that will reach the chosen WS from the LG due to this assignment. (In actual implementation, the bin values are periodically reduced by an amount equal to the minimum of all bin values so that the bin values will not become too large.)

3.2 Algorithms using load information on Web servers

A different approach to the scheduling issue is to allow the DNS to make decisions based on some information about the load of the Web servers. This is often used in conventional load balancing approach [25, 23, 24]. Wang and Morris have identified seven levels of information dependency, where the information of a higher level subsumes that of the lower level [25]. In our system, the lowest level may be represented by the *random* algorithm that does not require any information. For the next levels, we consider the following information ordered by increasing level: overloaded servers, the current load of WSs, the current and past load of WSs, the sequence of the servers chosen in the previous assignments.

Any of these kinds of information can be embodied in a scheduling strategy. Some of them could be maintained at the DNS itself (e.g., sequence of previous assignments). For the remaining ones, we assume that asynchronously or periodically the DNS receives the requested parameters from each WS. For algorithms requiring server load information, unless otherwise specified, we adopt as load information the *utilization* that each WS evaluates every 15 seconds for the past 15 second duration and transmits to the DNS. More extensive information either is not easily accessible in a Web-server system or is not useful for DNS resolution purposes. In particular, we propose and compare five algorithms which are representative examples of the performance achievable by the policies based on server information. Various other algorithms have been tested, but their results are not reported because their performance is equivalent or poorer than those shown here.

Asynchronous alarms. This is the lowest information dependency level that can come from a WS to the DNS, i.e. a binary state information indicating whether a WS is overloaded or not. Each WS periodically evaluates its own utilization and sends *asynchronous alarms* to the DNS to signal the beginning and end of a heavily loaded state, respectively. This information itself is not sufficient to define a scheduling policy, but it can be usefully combined with other algorithms (see Section 3.3.1).

Present load information. This policy, similarly to the classic Join-the-Shortest-Queue algorithm, considers only the lastest load information in deciding the assignments. In this paper, we show the performance of the **Least Utilized Node** (**LUN**) policy that assigns the requests to the WS having the lowest utilization, based on the most recent load information that the DNS receives from Web servers. Recall that the utilization is measured for a 15 second interval.

Past and present load information. The classic approach of making scheduling decisions on the

basis of present and past information motivates these algorithms.

Lowest Utilization (LU). The first version is a naive policy that estimates the load of a server by evaluating its utilization over a long period. To clearly differentiate this algorithm from LUN and successive policies, we consider the last hour as the interval for the utilization estimate for LU. As in LUN, the DNS assigns the requests to the WS that has the lowest utilization.

Lowest among Past and Present Utilizations (LPPU). In this case, the DNS attempts to estimate the future load by assigning lower weights to less recent measures. In this paper, we present the LPPU policy that uses the five most recent utilization samples as the measure of the WS load. Each value is combined with a weight obtained by a decay distribution. We use $pput_i = \sum_{j=1}^{5}(1/G)\rho_i(j)e^{-j/2}$, where $\rho_i(j)$ for $j \in \{1,\ldots,5\}$ are the utilization samples of WS_i evaluated in the past five intervals of 15 seconds each, and $G = \sum_{j=1}^{5} e^{-j/2}$ is the normalizing factor. The DNS selects the WS that has the lowest weighted utilization (*pput*) value. Experiments carried out with different parameter values did not significantly change the LPPU results.

Sequence of previous assignments. The idea for this class of policies comes from the observation that a DNS address resolution has an impact over a period of time. In particular, we propose a variation of the naive LUN and LPPU above algorithms (referred as **LUN-modified** and **LPPU-modified**, respectively). In the modified version, these policies look for the first and second minimum among the WS utilization samples. If the first minimum corresponds to the same WS of the previous assignment, the other WS is chosen. Note that although these versions increase the information dependency level, they do not increase the communications overhead because the sequence of Web servers chosen in the previous assignments are already available at the DNS. One could propose different implementations in which even the third or fourth minimum utilization samples are considered (i.e, the third minimum utilized WS is selected if the first two were selected in the previous two assignments). However, in the simulations, no significant changes to performance were observed by considering the third or fourth minimum utilized WS; therefore, these possible extensions are omitted from further consideration.

3.3 Algorithms combining domain and server information

The last step in the design of the scheduling policies is to allow the DNS to make decisions based on information on LG as well as WS loads. We differentiate between algorithms using very little information such as asynchronous alarms from heavily loaded servers, and policies needing accurate Web server load information.

3.3.1 Algorithms using alarm messages from heavily loaded servers

The implementation of both RR2 and DAL algorithms is relatively simple and does not require tracking or monitoring of the actual load condition on the WSs. On the other hand, the simple information on which they base their decisions does not always reflect the actual WS loads. The experimental results in Section 5 will show that this sometimes causes a WS to become overloaded even if its estimated load (e.g., bin level in DAL) is not high. A potentially better approach is to combine the previous strategies based on LG information with some additional information about the actual load. In this section, we focus on algorithms that take into consideration a simple state information indicating whether a WS is heavily loaded or not. Each WS periodically evaluates its own utilization and it is allowed to send *asynchronous alarms* to the DNS, signaling the beginning and end of risk of an overloading phase, respectively.

Thus the scheduling algorithms can base their decisions on some feedback information about the actual load. This alarm takes precedence over any load estimation or regular scheduling. Typically, the overloaded WS is taken off the pool of *active servers* and is not considered in any assignment until the end of the overloading situation signaled by another message. If all the WSs are considered *inactive*, a random assignment is forced. We analyze three variations of this approach with an asynchronous alarm that can be combined with any strategy using domain information such as RR, RR2 and DAL.

Single threshold (Thr1). The DNS can distribute client requests among all WSs using one of the algorithms previously described, unless it receives an alarm message from some of the Web servers indicating that its *utilization* has exceeded a *critical load threshold* T_{OUT}. In such a case, the DNS excludes the overloaded WS from further assignment of any requests until its utilization returns under the T_{OUT} threshold.

Double threshold (Thr2). This algorithm is similar to the Thr1 strategy. However, Thr2 tends to delay the re-activation of an overloaded WS by using a load threshold T_{OUT} to determine the exit from the active pool, and another (lower) threshold T_{IN} to allow for the re-activation. The rationale for using two load thresholds is as follows. With a single threshold, if we set it too low, many servers will be considered *inactive* even when it is not necessary. Conversely, if we choose a high threshold, the re-activation may be premature, thereby causing new overloading situations for the just re-activated WS. For this reason, the Thr2 strategy, that uses a relatively high threshold to trigger an exit from the active pool and a lower threshold for re-activation, provides an anti-thrashing mechanism that partially resembles the second algorithm to determine process migration in [24]. Multiple thresholds to define the load status of a node and job re-direction were also proposed in [23].

Temporal threshold (ThrT). This method introduces another anti-thrashing mechanism to account for the fact that a delay is present between the exit of a WS from the active pool for scheduling and its return to the active pool. ThrT uses a single threshold and excludes a WS when its utilization exceeds that threshold. However, the re-activation is not based on the utilization level and a new message to the DNS, but on a time estimation (provided by

DNS) that is deemed necessary for a WS to complete most of the requests due to the past WS address assignments. The basic idea behind this algorithm is similar to the threshold with delay that motivated the third algorithm in [24].

3.3.2 Algorithms using additional information about the server load

A further step in the optimization of the DNS scheduler would seem to allow the DNS to make decisions based on more precise information about the actual load of the WSs. For this reason, we propose some variations of the DAL policy that combine the domain information with some information about the load status of the Web servers.

DAL-set bin to top (DAL-ST). This algorithm is a simple modification of the DAL policy. However, DAL-ST uses the asynchronous alarm messages from the WSs differently from DAL-Thr1 and DAL-Thr2. The idea is that every time the utilization of a WS exceeds the given threshold, its bin value must be set at least equal to the current highest bin value.

DAL-set bin to actual number of requests (DAL-AN). This algorithm works similar to DAL with the exception that the DNS tries to calibrate the bin value of each WS as follows. Every TTL seconds, each WS sends to the DNS the number of requests that it has received in that period. Hence, the DNS can modify the bin value by subtracting (from the estimated load weight) the actual number of requests that have been served in the last TTL interval. Analogously to DAL, at the arrival of an address request, the DNS chooses the WS with the lowest bin value.

Minimum Residual Load (MRL). This algorithm is a variation of DAL that combines domain with server information. Analogous to the previous algorithm, MRL tracks the hidden load weight of each LG. In addition, the DNS maintains an *assignment table* containing all the assignments and their times of occurrence. Let l_j is the average session length of a client of LG_j. After a period of TTL+l_j, the effect of the assignment is expected to expire, i.e. no more requests will be sent to LG_j due to this assignment. The entry for that assignment can hence be deleted from the assignment table. At the arrival of an address resolution request, the DNS evaluates the expected number of residual requests that each WS will receive based on the previous assignments, and chooses the WS with the minimum number of residual requests. Here we assume the subsequent requests spread evenly across the period of TTL+l_j when the name-to-address mapping is in effect. Hence, at the time t_{now}, the DNS computes:

$$\min_{i=1,\ldots,N} \left\{ \sum_{j \in \{LG\} \rightarrow WS_i} \sum_{k} \left[w_j \left(t_j(i,k) + TTL + l_j - t_{now} \right)_+ \right] / (TTL + l_j) \right\}$$

where w_j is the hidden load weight of LG_j, and $t_j(i,k)$ is the time of the assignment of the $k-$th address resolution request coming from LG_j to WS_i in the mapping table. The $(x)_+$ notation denotes that $(x)_+ = x$ if $x > 0$ and $(x)_+ = 0$ if $x \leq 0$, i.e. that only the positive terms are considered in the internal sum. As mentioned above, a request entry is removed from the assignment table when the corresponding term $(t_j(i,k) + TTL + l_j - t_{now})$ is detected to be

Used information	*Algorithm*	*Definition*
Domain	**RR**	Round-Robin
	RR2	Round-Robin 2-tier
	DAL	Dynamically Accumulated Load
WS load	**LUN**	Least Utilized Node
	LU	Lowest Utilization
	LPPU	Lowest among Past and Present Utilizations
	-modified	The second minimum is chosen
Domain and WS load	**DAL-ST**	DAL where bin is set to top
	DAL-AN	DAL where bin is set to the actual number of requests
	MRL	Minimum Residual Load
WS alarm	**-Thr1**	Single alarm threshold: $T_{OUT} = T_{IN}$
	-Thr2	Double alarm threshold: $T_{OUT} > T_{IN}$
	-ThrT	Temporal threshold: T_{OUT} (utilization), T_{IN} (time)

Table 1: DNS scheduling algorithms classified on the basis of the adopted system information (first column). The *WS alarm* policies are always combined with one of the previous algorithms.

negative since no more residual load is expected to remain from this assignment. We note that since the average session lengths l_j are not readily available at the DNS but they can be estimated at the Web servers (e.g. using the cookie mechanism to track user sessions), the implementation of MRL in a real environment would require periodic messages from the servers to the DNS. Certainly, we expect l_j to be rather stable over time so the frequency of exchanges should be quite low. This is in contrast to the load information of the WS which requires frequent updates. MRL can also be combined with asynchronous alarm messages. We will refer to this last policy as MRL-Thr.

Table 1 contains acronyms and definitions of all presented algorithms.

4 Parameters and methods of evaluation

In this section we present the parameters with their base values and distributions that characterize the entire system and the DNS scheduling algorithms. The distributed Web-server system is composed of N servers with the same capacity. The average load of the entire system is maintained at 0.667 that is, 2/3 of the system capacity. Since the servers are mirrored, the page requests from the clients can be serviced at any WS. The number of page requests per session and the inter-request time are exponentially distributed [2]. Cunha *et al.* have measured roughly seven different hits per page [11]. More recent analyses indicate that this mean number is increasing. Therefore, we have modeled the number of hits per page as obtained from a uniform distribution in the interval [5–15]. The service time for each hit is assumed to be exponentially distributed with a mean of 4.5 msec.

The network consists of a set of intermediate name servers (INS) that cache the name-to-address mapping for a TTL period. The mean number of INSs that an address resolution request has to cross from its LG to the DNS is chosen equal to the mean number of hops shown in [4]. This function

Category	*Parameter*	*Values (default)*
Web-server system	Number of servers	4–9 (7)
	Average utilization	0.5–0.8 (0.6667)
LG	Number	10–100 (20)
	Client distribution among LG	Zipf (pure Zipf)
		geometric ($p = 0.3$)
	TTL (seconds)	0–360 (240)
Client	Number	1500-5000 (1500)
	Web page requests per session	exponential (mean 20)
	Hits per Web page	uniform [5–15]
	Hit service time (milliseconds)	exponential (mean 4.5)
	Inter-request time (seconds)	exponential (mean 15)

Table 2: Parameters of the system.

is similar to a double bell Gaussian distribution with peaks around 2 and 16 corresponding to *local* and *remote* requests, respectively.

The clients are typically partitioned among the domains based on a pure Zipf's distribution, as discussed in Section 2.3. However, we also carried out other experiments to evaluate the sensitivity of the DNS scheduling policies to other distributions, and we also varied the percentage of single user domains, i.e. *singleton* domains. Table 2 summarizes the system and workload parameters that we used in our simulations. The algorithms using WS load information evaluate the utilization of each server every 15 seconds and transmit this value to the DNS with the same frequency. The algorithms using WS alarms evaluate the utilization of each server every 8 seconds and send a signal to the DNS only if the alarm threshold is crossed. In particular, Thr1, ThrT and DAL-ST use as default $T_{OUT} = 0.75$, while Thr2 uses $T_{OUT} = 0.80$ and $T_{IN} = 0.70$. Unless otherwise specified, the figures of the experimental results refer to the default values.

The *performance stability* is the main concern for the distributed Web-server system that is subject to non-uniform bursts of arrivals of which only a small percentage is directly controllable by the DNS. Therefore, we are more interested in investigating the impact of the DNS algorithms on avoiding overload at any of the servers than on equalizing the system workload. The reason for not adopting traditional metrics, such as the standard deviation of cumulative utilizations which is typical of load balancing studies, is that we analyze the performance of a transactional distributed system with independent nodes, and not a computing platform with inter-related distributed tasks.

A good index to measure the load level and overloaded instances is the *utilization* of the Web servers. In particular, looking at the most utilized WS among all WSs, we can deduce whether the Web-server system is overloaded or not. Hence, the performance of the various scheduling policies is evaluated by focusing on the maximum WS utilizations observed during the simulation runs. That is to say, we focus on the utilization level of the server with the maximum utilization among all servers at each time instant. The server with the maximum utilization changes over time. If the maximum utilization at an instant is low, it means that no server is overloaded at that time. By tracking the period of time the maximum utilization is above or below a certain threshold, we can get an indication of how well the distributed system is running. For example, assume three

servers. If their utilizations are 60%, 75% and 63%, respectively, at time t_1 and 90%, 66% and 42%, respectively, at time t_2, the maximum utilization at t_1 is 75% and that at t_2 is 90%. With a maximum utilization of 90%, the distributed system has a load balancing problem at t_2. By examining the histogram or cumulative frequency of the maximum utilization, we can determine the percentage of time for which at least one of the servers is critically loaded (say exceeding 90% utilized).

Hence we use the *cumulative frequency* of the maximum utilization among all Web servers as the performance criteria. More specifically, at the occurrence of an observation or sampling instant, we compute the utilization for each Web server in the previous observation interval. Then we sort the list of server utilizations into decreasing order and track the cumulative frequencies for the maximum, the second maximum and the third maximum utilizations. Note that usually all the servers, even if with different proportions, contribute to this maximum utilization during the entire simulation, i.e. each server can and indeed has become the maximally utilized server for some period of time during the simulation. Finally, we plot the cumulative frequency of the maximum utilizations observed after the transient state of the simulator. As shown in detail in Section 5, with most promising scheduling policies (RR2, DAL, MRL) usually at most one server is overloaded while the second highest utilization rarely goes beyond 0.9 (as shown in Figure 11) and the third never exceeds 0.9 (as shown in Figure 12); thus, the cumulative frequency of the maximum utilization gives a precise estimate of the ability of a scheduling policy to minimize overload situations. For example, consider the point $(0.78, 0.9)$ on the RR(Uniform) curve in Figure 3. It indicates that with probability 0.9 the utilization of the busiest Web server is below 0.78.

Since the average system utilization is set to 0.667, the distribution of the maximum utilization of a *perfect* policy (always maintaining a utilization of 0.667 at each WS) should be a step function which goes from 0 to 1 at a utilization of 0.667. Since this result is unrealistic, in Section 5 we consider as *ideal* algorithm, the *DNS full control* policy, i.e. the RR for the case with TTL=0. Although this algorithm has worse performance than a perfect policy, it can be considered an ideal target because, under a realistic scenario of non-uniform client distributions and TTL$\gg$ 0, no scheduling algorithm can achieve the *DNS full control* performance.

When we evaluate the sensitivity of the scheduling algorithm performance to other system parameters such as TTL and percentage of singleton domains, we adopt a different metric. Indeed, it is preferable to show the 96-th percentile of the maximum utilization that is, the $Prob(maxUtilization < 0.96)$, which we use as probability indicator that no server of the Web-server system is overloaded (exceeding 96% utilized).

The simulators, based on the Independent Replication Method, were implemented using the CSIM package [22]. Each value is the result of five simulation runs with different seeds, where each run is for six hours of the Web-server system activity. For all simulation results shown in the next section, confidence intervals were estimated, and the 95% confidence interval was estimated to be within 4% of the mean.

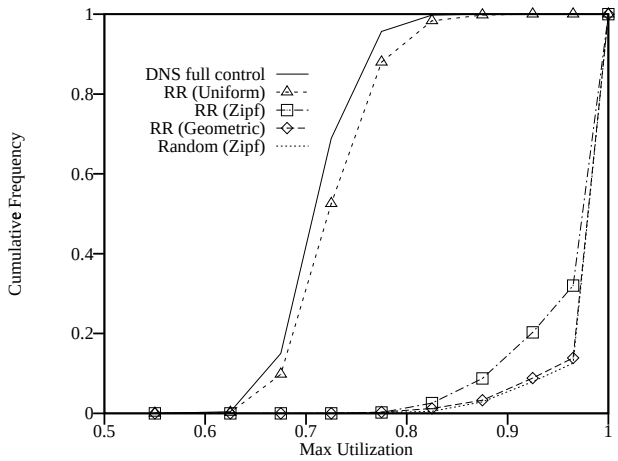

Figure 3: Distribution of maximum server utilization by RR scheduling policies.

5 Experimental Results

For the performance evaluation of the proposed scheduling algorithms, we carried out a large number of experiments. Due to space limitations, only a fraction is presented here. The goal is to measure how effectively the DNS scheduling algorithms, that control only a very small percentage of the address resolution requests, can minimize overload situations of a distributed Web-server system.

5.1 Algorithms using domain information

In the first set of experiments we focus on DNS algorithms that use information about the origin of the client requests as the basis for making scheduling decisions.

Figure 3 shows the performance of the RR algorithm in an environment where the clients are uniformly distributed among the domains as compared to that of the RR where the number of clients behind each domain is modeled by a pure Zipf's distribution. We assume a distributed system with 20 domains and TTL set to 240 seconds. This figure provides one of the principal motivating factors for this study. While the RR policy under uniform distribution performs close to the *ideal* policy that gives full control to the DNS for requests reaching the Web-server system (*DNS full control* is obtained for a TTL=0), under the more realistic case of a Zipf's distribution, Figure 3 shows that $Prob(maxUtilization < 0.96) \approx 0.30$. This means that RR applied to a scenario with a skewed client distribution causes at least one WS to be highly overloaded (0.96 utilized or above) for almost 70% of the time. The performance is even worse when clients are geometrically distributed. Also shown in this figure is the *random* policy, for a Zipf's distribution of clients behind a LG, where a Web server is randomly chosen for each assignment. Not surprisingly, this has the worst performance.

Figure 4 compares the behavior of the three algorithms that only use LG information. Both DAL and RR2 exhibit much better performance than RR, while the DAL policy performs the best.

Again a system with 20 domains and TTL set to 240 seconds is assumed.

The performance improvement achieved by DAL and RR2 with respect to RR-DNS is again shown by Figure 5 for a scenario in which the TTL period chosen by DNS is reduced to 150 seconds. Under this more controlled environment, where a higher number of address resolution requests reaches the DNS, both DAL and RR2 exhibit performance with a convex shape which is similar to the ideal curve. The relative order among the strategies changes a little, and RR2 now performs a bit better than DAL. The main result shown in Figure 5 is that, when the DNS controls even a slightly higher percentage of requests (6–7% as compared to 2–3% of Figure 4), even the strategies that use only domain information achieve a significant improvement as compared to the RR policy. The probability that no WS is overloaded is more than 90% for DAL and RR2, and less than 60% for RR. However, in the more realistic scenario shown in Figure 4 where TTL is set to 240 seconds, the probability of having an unbalanced server is still high for any policy that only uses information on the origin of the client requests, and does not use server load information.

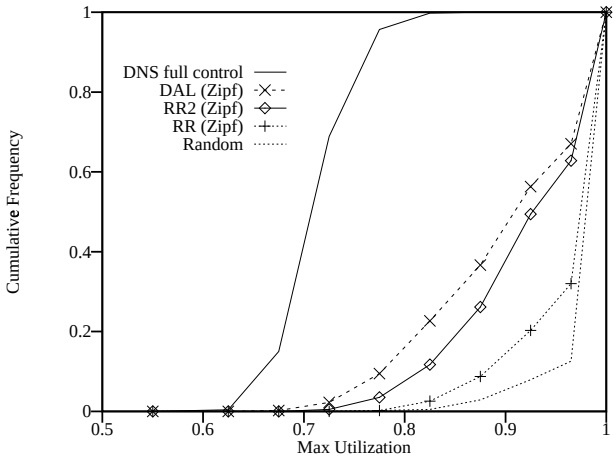

Figure 4: Performance of scheduling policies with domain information (TTL=240).

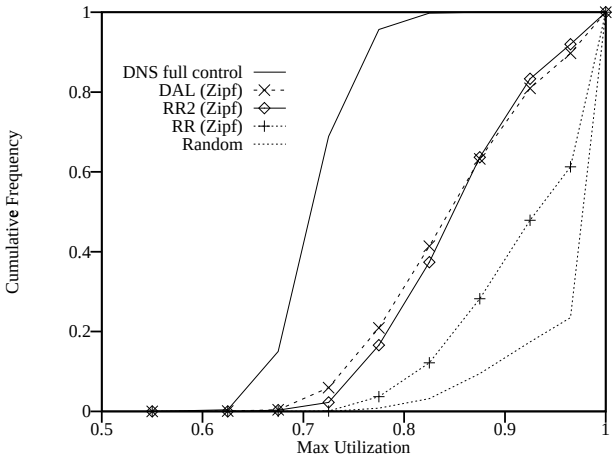

Figure 5: Performance of scheduling policies with domain information (TTL=150).

5.2 Algorithms using server load information

We now evaluate the performance of the algorithms that can base their scheduling decisions on the actual load of the WSs. Figure 6 shows the performance of all the policies discussed in Section 3.2 in order to investigate which information about server load is most useful. The results indicates that it is preferable to make scheduling decisions based on the most recent load information only.

The LU policy, which uses more past (or historical) information, has very poor performance (even worse than the random policy), while the LPPU policy, that gives a minor weight to the less recent information, is much better than LU. However, the best policy is LUN that considers only the lastest load information. In particular, LUN-naive is better than LPPU-naive, and LUN-modified is better than LPPU-modified. Not surprisingly, the modified versions of the last two policies, which use information on the second minimum, have better performance than their naive

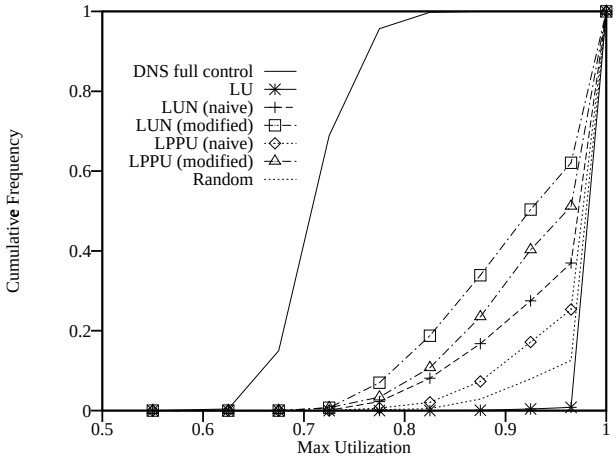

Figure 6: Performance of scheduling policies with accurate load information from server.

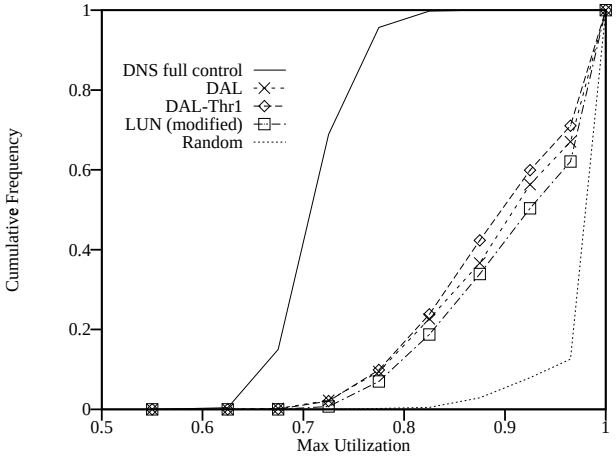

Figure 7: Comparison of scheduling policies with synchronous and asynchronous server information.

counterparts.

The reason for the ineffectiveness of policies based on more historical load information can be attributed to the dynamics of the distributed Web-server system. Due to the large variability of the LGs or clients assigned to a WS, the server load information becomes obsolete quickly and is poorly correlated with future WS load.

As further confirmation of this result, we observed (not shown here) that the LUN-modified policy has increasingly better performance with a higher frequency of sending WS load information to the DNS. However, this improvement is achieved at the price of extra computation and communications overheads due to the higher number of messages exchanged between the WSs and DNS. In traditional parallel/distributed systems, this cost is often worthwhile because a centralized scheduler that tracks the actual load of all the servers usually leads to the best performance. This is not the case for the distributed Web-server system under consideration, because other policies, such as DAL, achieve even better results than LUN-modified with lower communications cost as shown in Figure 7. The performance is even better, if DAL is combined with an alarm message (DAL-Thr1) that does not impose a large overhead, as discussed in more detail in the next section. Therefore, we can conclude that in the distributed Web-server system, as in [14], the threshold policy using a small amount of state information (inexpensive both from the computational and the communications point of view) provides the most substantial performance improvement. This motivates the research for policies that combine client and server information.

5.3 Algorithms combining domain and server information

In this section we evaluate the performance of policies that combine domain with server information, without requiring the frequent exchange of messages as in the LUN policy. Specifically, we focus on policies using the overload alarm, and the MRL policy that also requires an estimate of the mean time of a client session. This information requires low communications overheads, because

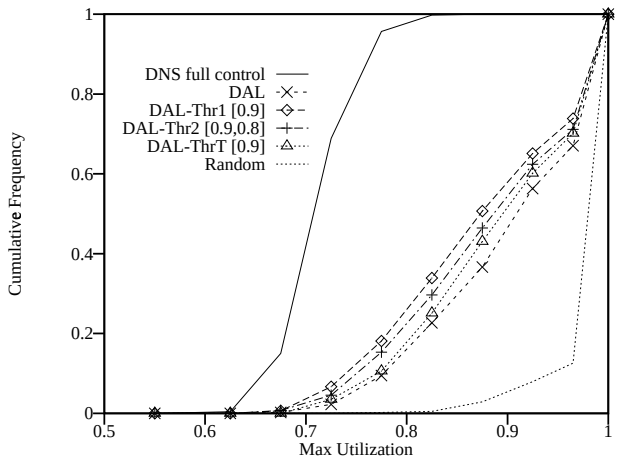

Figure 8: Performance of DAL policy with various kinds of alarm messages from WS.

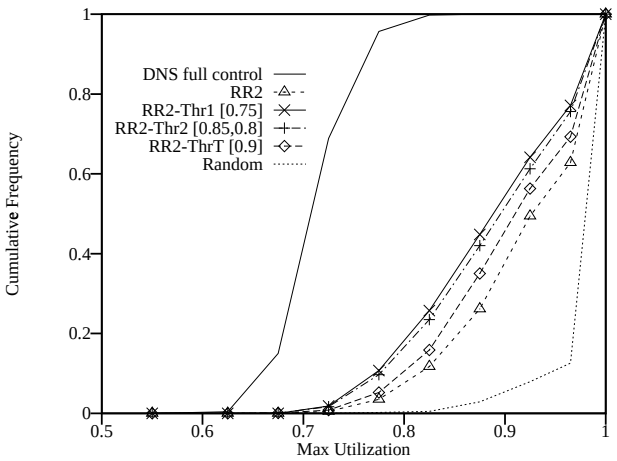

Figure 9: Performance of RR2 policy with various kinds of alarm messages from WS.

the former is implemented through asynchronous messages, while the latter is synchronous but the period can be large (i.e. of the order of minutes or tens of minutes), since the information is typically stable.

We compare the results of the three feedback strategies (Thr1, Thr2 and ThrT) described in Section 3.3.1 that combine asynchronous WS alarms with the DAL and RR2 algorithms. For each threshold policy, the results shown in Figures 8 and 9 refer to the best set of parameters found through a very large set of experiments. Assuming a 0.05 difference in T_{OUT} and T_{IN}, the best T_{OUT} and T_{IN} values for this scenario are between square brackets.

We see that simply tracking the heavily loaded servers improves the performance. In particular, ThrT does not improve the performance much as compared to the algorithms without feedback, while more consistent improvements can be achieved by using either Thr1 or Thr2. This demonstrates that a temporal threshold, such as ThrT, is not preferable to a quantitative threshold in deciding when a WS should re-enter the set of active servers.

The differences between Thr1 and Thr2 are not consistent. In particular, DAL and RR2 with two thresholds do not perform better than the corresponding algorithms with one threshold. Therefore, our preference is for the strategies with a single threshold T_{OUT}. This is for two main reasons: the strategies with a single threshold achieve a performance which is comparable to the policies with two thresholds; they have one less parameter, thereby simplifying the tuning of the scheduling algorithm in a dynamic scenario.

This section shows that all the algorithms with any alarm feedback from WSs to the DNS have better performance than their non-feedback counterparts. This result is important because this performance improvement is achieved at little increase of the communications overhead.

To understand usefulness of different pieces of information, Figure 10 compares various versions of the DAL policy against the MRL algorithm. First, we can observe that the pure DAL is much better than DAL-AN. DAL-ST, not shown for the clarity of exposition, has performance similar to

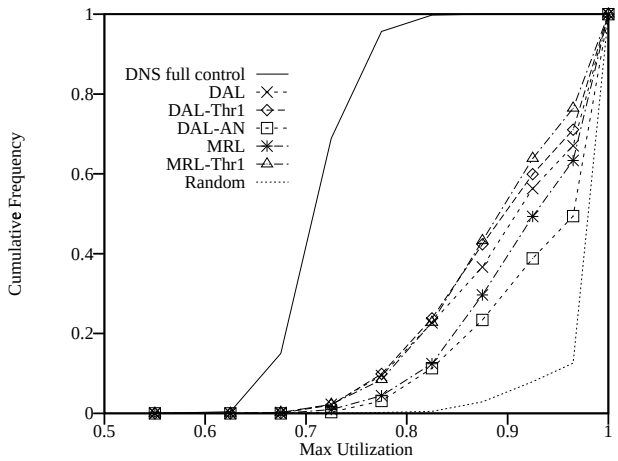

Figure 10: Performance of DAL and MRL policy combined with various server information.

DAL-AN. DAL is even better than MRL, while DAL-Thr1 is the best DAL policy. Hence, we can conclude that, among various server information, the alarm signal from heavily loaded servers is the most effective. Furthermore, if we compare MRL-Thr1 and DAL-Thr1, it is the MRL-Thr1 policy that achieves the best performance. This swap in order with the presence of asynchronous alarms, as compared with MRL and DAL, can also be observed when we compare the results in Figure 8 and 9. Without alarm signals, DAL is better than RR2, while with alarm feedback, RR2-Thr1 performs better than DAL-Thr1.

Another interesting result is the following. Although no policy guarantees that the Web-server system is never overloaded, Figures 11 and 12 indicate that, for the proposed policies these overloading events occur very rarely in more than one server. Specifically, Figure 11 and Figure 12 show the maximum utilization of the second and third highest utilized Web server, respectively. These figures provide indication of whether more than one server is overloaded for RR, DAL, RR2-Thr1 and DAL-Thr1. Quite analogous results are obtained for all other proposed policies. Indeed, the second highest utilizations (Figure 11) of the proposed schemes are always under 0.85 that is, the second maximum utilizations are less than 0.85 with cumulative frequency or probability one. RR is the only policy showing frequent overload situations for more than one server: its second highest utilization is considerably below the other curves. In addition, the third highest utilizations (Figure 12) of the proposed schemes are around or below the average utilization 0.667. This result also confirms that focusing on the most loaded WS, instead of looking at the entire system unbalance, is a good metric to evaluate and compare the performance of DNS scheduling algorithms.

Figure 13 summarizes the performance of the policies that achieved best results for various scenarios of which only a part is shown in this paper. MRL-Thr1 and RR2-Thr1 can be considered the best policies with slight differences depending on the scenario. DAL-Thr1 is typically the third. We give the following explanation for this order. Although DAL attempts to make a more precise estimation of the load than RR2, the estimate is too crude to improve WS assignments. Recall that its load estimates are derived from average values of the hidden load weight that may not be valid for specific instances. On the other hand, a policy such as MRL, that also uses

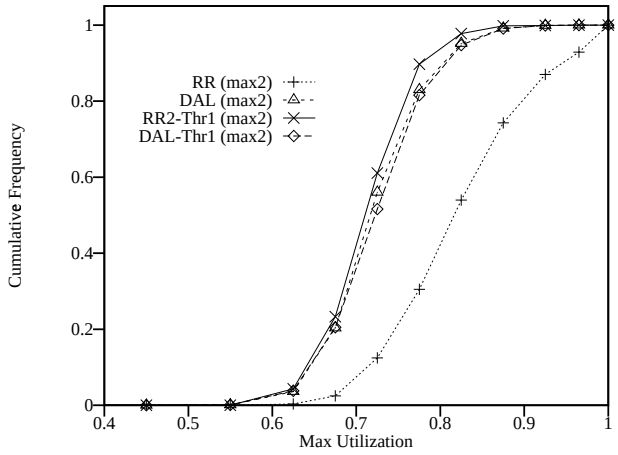

Figure 11: Distributions of the second highest server utilizations by various DNS scheduling policies.

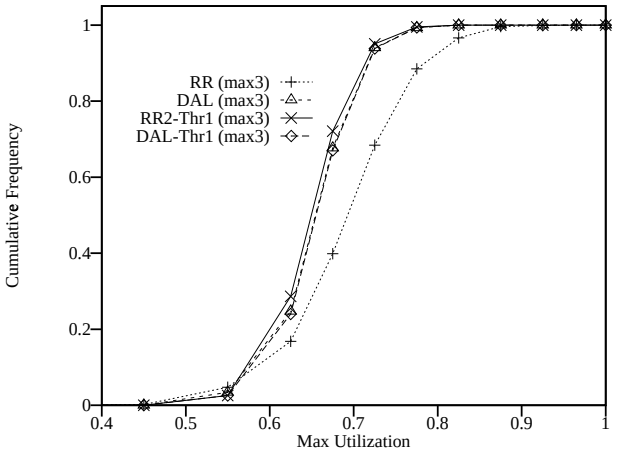

Figure 12: Distributions of the third highest server utilizations by various DNS scheduling policies.

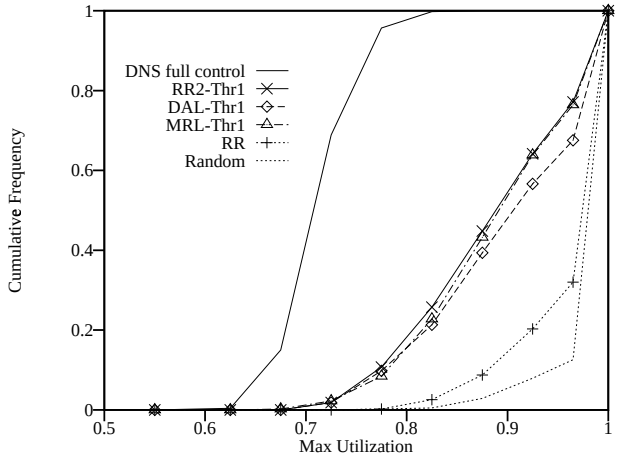

Figure 13: Performance of best scheduling policies combined with one-threshold alarm messages (TTL=240).

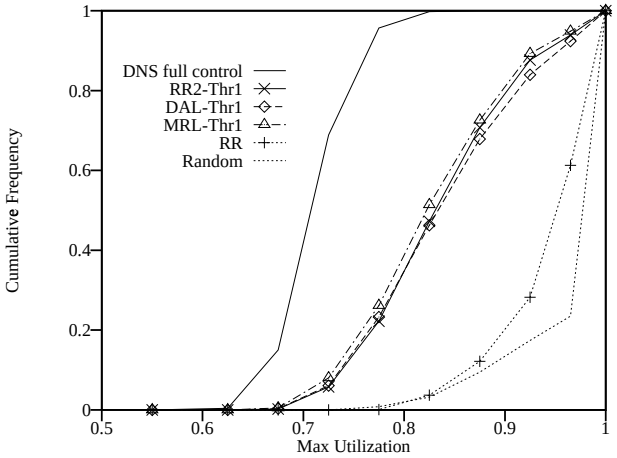

Figure 14: Performance of best scheduling policies combined with one-threshold alarm messages (TTL=150).

information about the residual active requests, seems to perform slightly better in terms of reducing the frequency of overload situations. Nevertheless, all these proposed policies considerably improve the performance of the existing RR-DNS. The improvements achieved by these algorithms become even more substantial when the DNS can control more requests through a reduction of the TTL value. Figure 14 shows that their performance now gets closer to that of the ideal policy.

5.4 Sensitivity of DNS scheduling algorithms

We now evaluate the sensitivity of the best performing policies to various system parameters. The performance metrics used is the 96th percentile of the cumulative frequency of the highest utilized WS. In other words, the y-axis reports the probability that no server of the distributed Web-server system is overloaded (i.e. more than 96% utilized).

In Figure 15 we examine the sensitivity to the TTL that is, the period during which the name servers cache a name-to-address mapping. The ordering of the policies by performance is essentially the same, with change in the TTL value. The strategies with asynchronous alarms show similar results that are considerably better than the RR-DNS policy across the range of reasonable TTL values. The percentage of requests controlled by the DNS decreases very rapidly as the TTL increases. However, as already observed in Section 2, we cannot solve the problem of limited DNS control by choosing a TTL value close to 0. First of all, this would cause the DNS to become a bottleneck and increase the number of messages in the network. Moreover, in order to reduce this traffic, most name servers set their own TTL values if the DNS proposed value is considered too low.

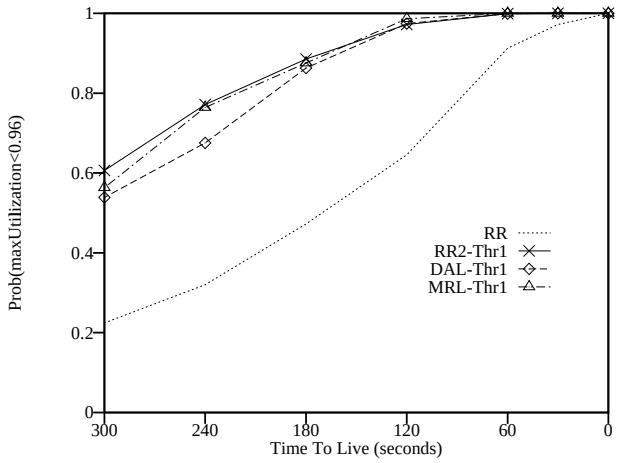

Figure 15: Sensitivity to the caching period (TTL) of a name-to-address mapping.

We now examine the sensitivity of the results to the distribution of clients among the domains. Figure 16 shows the sensitivity of the proposed policies to the mean number of connected domains, while the number of clients is fixed at 1500. This figure shows that there is a slight improvement for an increasing number of LGs. For the RR2-Thr1 and MRL-Thr1 policies, the probability that no server is overloaded is around 90% for more than 40 domains. The DAL policy has slightly

worse performance; however, the probability that no server is overloaded remains over 0.8 for more than 40 domains, which is significantly better than the 0.68 value achieved for the default value (20) of the number of domains. The RR-DNS policy, by comparison, improves marginally with an increasing number of LGs, but remains much worse than the proposed policies.

Another interesting aspect is the sensitivity of the scheduling policies to the percentage of requests coming from *singleton* domains as shown in Figure 17, where a singleton domain consists of only one client. We added to the system a sizable number of singletons to model a situation which is not provided by the long tail of the pure Zipf distribution. Indeed, the clients of the singleton domains rarely find in the cache of their LNS a valid name-to-address mapping. Therefore, they are forced to send their address resolution requests to the DNS, unless they find a valid one cached in some INS. Although each of these clients generates a small fraction of the workload on the Web-server system, considering more singleton domains increases the control of the DNS on the requests. This explains the better performance shown by all the proposed scheduling strategies when the percentage of requests coming from singleton domains increases. Note however, that the ordering between the policies stays the same as the percentage of singletons is varied. The RR-DNS policy actually has worse performance, as the percentage of singleton domains increases; an explanation for this behavior is that, in a round of the RR-DNS policy, requests from singleton domains can lead to very small loads on the server assigned to them, while a high load can result from nodes assigned to larger domains, giving rise to a larger variance in the load allocated per round.

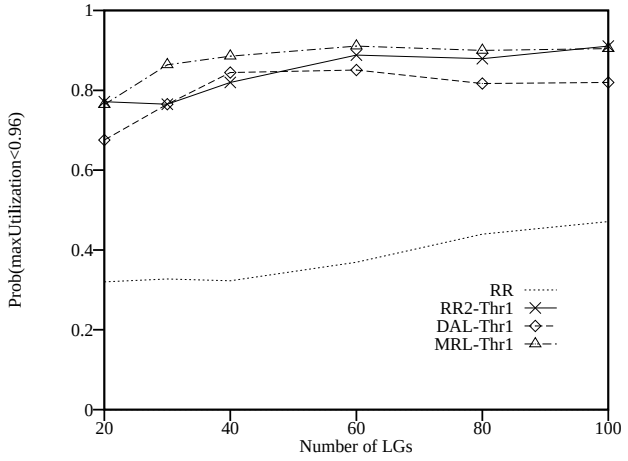

Figure 16: Sensitivity to the number of Local Gateways (LG).

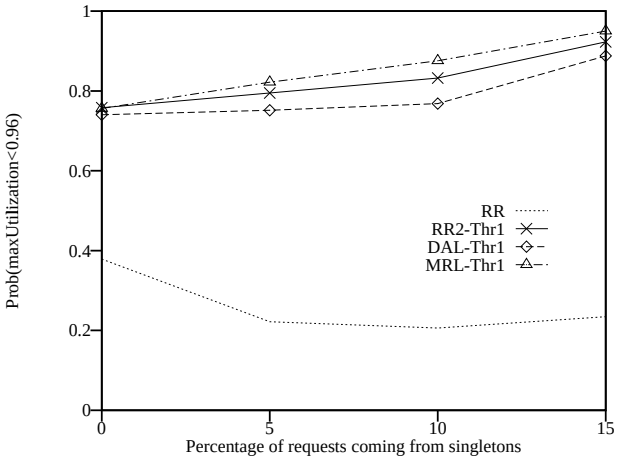

Figure 17: Sensitivity to the percentage of requests coming from singleton domains.

In the last set of experiments we evaluate the impact of the distribution of client requests on the performance of the best three scheduling policies. Figures 18, 19 and 20 refer to the RR2-Thr1, DAL-Thr1 and MRL-Thr1 policies, respectively. As shown in these figures, the performance under the geometric distribution (with $p = 0.3$) is similar to that under the pure-Zipf distribution.

In addition, these figures show how the results improve as the client distribution resembles the uniform distribution and the skew reduces. In particular, the DAL-Thr1 and MRL-Thr1 are

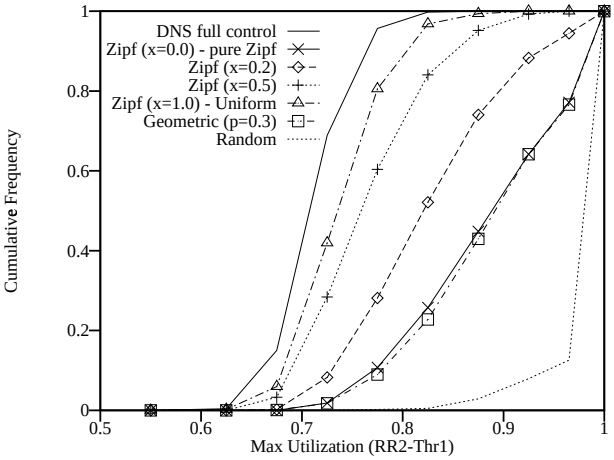

Figure 18: Sensitivity to geometric and various Zipf distributions of clients (*RR2-Thr1 policy*).

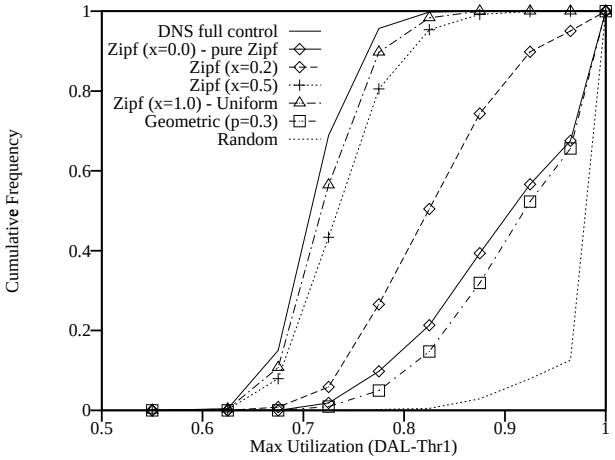

Figure 19: Sensitivity to geometric and various Zipf distributions of clients (*DAL-Thr1 policy*).

very close to the DNS full control curve even for Zipf with $x = 0.5$, while the RR2-Thr1 shows a somewhat smaller improvement when the client skew diminishes.

For a direct comparison of the three best policies under distributions different from the pure Zipf (already observed in Figure 13), Figure 21 shows the performance under geometric and Zipf distribution with parameter $p = 0.3$ and $x = 0.5$, respectively. For a geometric distribution, the relative order of these three policies remains unchanged with respect to a pure Zipf that is, MRL-Thr1 and RR2-Thr1 perform similarly and both better than DAL-Thr1. The relative order changes for a distribution with a lower skew, such as the Zipf with $x = 0.5$. Although all policies improve considerably to the extent that the Web-server system can be considered never overloaded, the RR2-Thr1 becomes the worst, which is consistent with the results shown in Figure 18.

Additional experiments (not shown due to space constraints) examined the sensitivity of the scheduling policies to different load models that could even include applications such as multimedia or electronic commerce. The common characteristics of these applications of Web servers is a likely growth in the number requests per user session (e.g. due to intensive Internet shopping at a site) and/or a longer service time that each hit requires at the server (possibly due to large multimedia objects or database access). The results indicated that the RR2-Thr1, MRL-Thr1 and DAL-Thr1 strategies are quite robust even for workloads that are very different from those considered earlier in the paper. In particular, RR2-Thr1 and MRL-Thr1 perform slightly better than DAL-Thr1. Analogous results are also obtained when increasing the mean inter-request time between hits.

We have shown in various scenarios that the performance of RR2-Thr1 and MRL-Thr1 is comparable, while the DAL-Thr1 is slightly worse. However, for a fair comparison, we also have to consider the implementation issues of these policies in an actual environment. The RR2-Thr1 policy has various merits: good and stable performance, easy implementation, and very low computational and communications overhead. Conversely, the modeling assumptions and hence results are more favorable than reality to algorithms such as LPPU, DAL and MRL. For example, the strategies

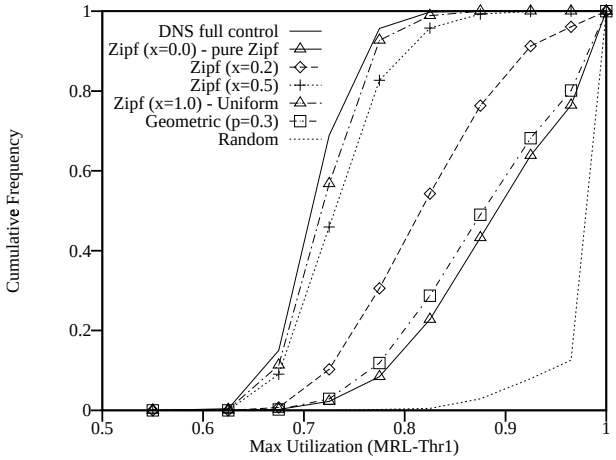

Figure 20: Sensitivity to geometric and various Zipf distributions of clients (*MRL-Thr1 policy*).

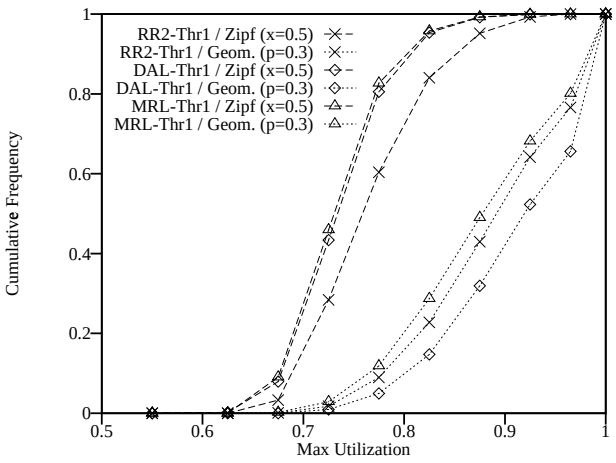

Figure 21: Comparison of Zipf ($x = 0.5$) with geometric distribution of clients.

with periodic messages from the servers here do not experience the random delays that would occur in the real network. Moreover, the algorithms that use the number of subsequent requests per name-to-address resolution (DAL) or the mean session time (MRL) in their load calculation could not estimate these parameters as accurately as in the simulation model, where the client parameters are obtained from theoretical distributions. In an actual environment, the analogous accuracy could be achieved only through frequent information exchanges from WSs to DNS. And this will increase the communications overhead of DAL and MRL policies.

6 Conclusions

In this paper, we adapted traditional scheduling algorithms to the Domain Name Server (DNS), proposed new policies, and examined their impact for various scenarios. The basic problem is that, typically, only a very small fraction of the client requests are mapped to a specific WS by the DNS, leading to little control by the DNS and consequent poor load balancing. With the goal of improving the basic round-robin scheme used by the DNS of some distributed multi-node Web servers, we considered three classes of algorithms that, through the use of additional state information, attempt to minimize the overloading situations of the Web servers. The first class uses the source domain of the clients in selecting the WS. This class of algorithms also estimates the number of subsequent client requests from each source following a name-to-address mapping, referred to as the hidden load weight. These algorithms attempt to prevent popular gateways from overloading specific WS nodes. The second class of algorithms uses load information on Web servers. We distinguish asynchronous alarms, that simply indicate heavily load conditions at WS, from detailed information on the WS load that require frequent message exchanges and cause more communications overhead. Finally, the third class of algorithms combines domain and server load information.

Extensive simulation results show the advantage of the strategies that take into account the domain of the requests and alarms from the servers. In particular, we found that classic algorithms such as *round-robin* and *least-loaded-server* are not adequate because current server load is not the best indicator of future server load due to the TTL effect. An initially unexpected result is that the best performance is achieved by algorithms that use only limited state information on whether a server is overloaded or not. On the other hand, detailed load information, especially the less recent load information, do not appear useful, and can often lead to false estimation about the future state. The most promising algorithms use asynchronous alarms (when the load at a WS crosses a threshold) combined with an estimation of the hidden load weight coming from each domain. This estimation can be simply used to partition the domains into two or more classes, with round-robin scheduling in each class (RR2 policy) separately, or can be combined with other information from Web servers such as the average session length to evaluate the residual load (MRL policy). All these algorithms can lead to much better load balancing than RR-DNS, while RR2-Thr1 is the simplest to implement. Extension to case of heterogeneous servers is considered in [9].

We note that the DNS policies presented in this paper could be applied to the recent work on Universal Resource Names *URN* [12]. This is a topic of further work.

References

[1] D. Andresen, T. Yang, V. Holmedahl, O.H. Ibarra, "SWEB: Toward a scalable World Wide Web server on multicomputers", *Proc. of 10th Int. Symp. on Parallel Processing* (IPPS'96), Honolulu, pp. 850–856, April 1996.

[2] M.F. Arli, C.L. Williamson, "Web server workload characterization: The search for invariants", *Proc. of ACM Sigmetrics '96*, Philadelphia, pp. 126–137, May 1996.

[3] M. Baentsch, L. Baum, G. Molter, "Enhancing the Web's infrastructure: From caching to replication", *IEEE Internet Computing*, vol. 1, no. 2, pp. 18–27, Mar.-Apr. 1997.

[4] A. Bestavros, "WWW traffic reduction and load balancing through server-based caching", *IEEE Concurrency*, vol. 5, no. 1, pp. 56–67, Jan.-Mar. 1997.

[5] H.-W. Braun, K.C. Claffy, "Web traffic characterization: an assessment of the impact of caching documents from NCSA's Web server", *Computer Networks and ISDN Systems*, vol. 28, pp. 37–51, 1995.

[6] V. Cardellini, M. Colajanni, P.S Yu, "Efficient state estimators for load control policies in scalable Web server clusters", *Proc. of 22nd IEEE Int. Computer Software and Application Conference* (COMPSAC'98), Vienna, Austria, Aug. 1998.

[7] T.L. Casavant, J.G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems", *IEEE Trans. Software Engineering*, vol. 14, no. 2, pp. 141–154, Feb. 1988.

[8] K.C. Claffy, H.-W. Braun, G.C. Polyzos, "Tracking long-term growth of the NSFNET", *Communications of the ACM*, vol. 37, no. 8, pp. 34–45, Aug. 1994.

[9] M. Colajanni, P.S. Yu, V. Cardelliini, "Dynamic load balancing in geographically distributed heterogeneous Web servers", *Proc. of 18th Intl. Conf. on Distributed Computing Systems*, Amsterdam, Netherland, May 1998.

[10] M. Crovella, A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes", *IEEE/ACM Trans. on Networking*, vol. 5 no. 6, pp. 835-846, Dec. 1997.

[11] C. Cunha, A. Bestavros, M. Crovella, "Characteristics of WWW client-based traces", Tech. Rep. BU-CS-95-010, Boston University, Computer Science Dept., April 1995.

[12] R. Daniel, M. Mealling, "Resolution of Uniform Resource Identifiers using the Domain Name System", Internet Draft draft-ietf-urn-naptr-05.txt, May 1997 (see also: http://www.acl.lanl.gov/URN/).

[13] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, "A scalable and highly available Web server", *Proc. of 41st IEEE Computer Society Int. Conf.* (COMPCON'96), Technologies for the Information Superhighway, pp. 85–92, Feb. 1996.

[14] D.L. Eager, E.D. Lazowska, J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems", *IEEE Trans. Software Engineering*, vol. SE-12, no. 5, pp. 662–675, May 1986.

[15] P. Krueger, N. G. Shivaratri, "Adaptive location policies for global scheduling", *IEEE Trans. Software Engineering*, vol. 20, no. 6, pp. 432–444, June 1994.

[16] T.T. Kwan, R. McGrath, D.A. Reed, "NCSA's World Wide Web server: Design and performance", *IEEE Computer*, vol. 28, no. 11, pp. 68–74, Nov. 1995.

[17] Y.-H. Liu, P. Dantzig, C.E. Wu, J. Challenger, L.M. Ni, "A distributed Web server and its performance analysis on multiple platforms", *Proc. of 16th Int. Conf. on Distributed Computing Systems* (ICDCS'96), Hong Kong, pp. 665–672, May 1996.

[18] R. McGrath, "What we do and don't know about the load on the NCSA WWW server", http://www.ncsa.uiuc.edu/InformationServers/Colloquia/28.Sep.94, Sept. 1994.

[19] J. Pitkow, "In search of reliable usage data on the WWW", *Proc. of 6th Int. World Wide Web Conference*, Santa Clara, CA, Apr. 1997.

[20] K. Ramamritham, J.A. Stankovic, W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements", *IEEE Trans. on Computers*, vol. C-38, no. 8, pp. 1110–1123, Aug. 1989.

[21] H. Schulzrinne, "World Wide Web: Whence, Whither, What next?", *IEEE Network*, pp. 10–17, Mar.-Apr. 1996.

[22] H. Schwetman, *CSIM17–User's Guide*, Mosquite Software Inc., 1994.

[23] K. G. Shin and Y.-C. Chang, "Load Sharing in distributed real-time systems with state-change broadcasts", *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1124-1142, Aug. 1989.

[24] J.A. Stankovic, "Simulations of three adaptive, decentralized controlled, job scheduling algorithms", *Computer Networks*, vol. 8, no. 3, pp. 199–217, June 1984.

[25] Y.T. Wang, R.J.T. Morris, "Load sharing in distributed systems", *IEEE Trans. on Computers*, vol. C-34, no. 3, pp. 204–217, Mar. 1985.

[26] G.K. Zipf, *Human Behaviour and the Principles of Least Effort*, Addison-Wesley, Cambridge, MA, 1949.