DNS Dispatching Algorithms with State Estimators for Scalable Web-server Clusters

V. Cardellini
University of Rome "Tor Vergata"
Roma, Italy 00133
cardellini@uniroma2.it

M. Colajanni
University of Modena
Modena, Italy 41100
colajanni@unimo.it

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

Abstract

Replication of information across a server cluster provides a promising way to support popular Web sites. However, a Web-server cluster requires some mechanism for the scheduling of requests to the most available server. One common approach is to use the local *Domain Name Server* (DNS) as a centralized dispatcher. The main problem is that WWW address caching mechanisms (although reducing network traffic) only let this DNS dispatcher control a very small fraction of the requests reaching the Web-server cluster. The non-uniformity of the load from different client domains, and the high variability of real Web workload introduce additional degrees of complexity to the load balancing issue. These characteristics make existing scheduling algorithms for traditional distributed systems not applicable to control the load of Web-server clusters and motivate the research on entirely new DNS policies that require some system state information. We analyze various DNS dispatching policies under realistic situations where state information needs to be estimated with low computation and communication overhead so as to be applicable to a Web cluster architecture. In a model of realistic scenarios for the Web cluster, a large set of simulation experiments shows that, by incorporating the proposed state estimators into the dispatching policies, the effectiveness of the DNS scheduling algorithms can improve substantially, in particular if compared to the results of DNS algorithms not using adequate state information.

Keywords: World Wide Web, Internet, Load balancing, Distributed system, Performance evaluation

1 INTRODUCTION

With the rapid growth of WWW traffic, most popular Web sites need to scale up their server capacities. One way is to provide a list of alternate, or equivalent mirrored servers at different locations. Letting each *Uniform Resource Locator* (URL) represent a single Web server has a number of disadvantages. First of all, the *mirrored server* choice is not transparent to the users. Moreover, it is hard to provide load balancing and fault-tolerance across the mirrored servers. The most promising approach to handle popular Web sites is to preserve a virtual single interface and to use a distributed architecture. Such an architecture is more scalable, fault-tolerant, and load balanced than a Web system based on independent mirrored sites. However, the achievement of these goals requires a coordination mechanism which is able to assign requests to the Web server that can offer the best service [Dias *et al.* 1996; Kwan *et al.* 1995; Colajanni *et al.* 1998b]. The assignment decision can be taken at the *IP-dispatcher* level or at the cluster *Domain Name System* (DNS) level. A round-robin DNS policy is implemented by the NCSA HTTP-server [Kwan *et al.* 1995] and the SWEB server [Andresen *et al.* 1996], while a randomization scheme is used by the Netscape browser [Mosedale *et al.* 1997]. A DNS policy that takes into account the geographical location of the client is implemented by the Cisco *DistributedDirector* [Cisco 1997a]. Other theoretical DNS scheduling policies that require some system state information are proposed in [Colajanni *et al.* 1998b; Colajanni *et al.* 1998a]. Various IP-dispatcher solutions are described in [Dias *et al.* 1996; Cisco 1997b; Anderson *et al.* 1996]. They have full control on the incoming requests, but they are typically applied to a *locally* clustered Web server. Moreover, the IP-dispatcher can become a bottleneck if the system is subject to heavy request load.

In this paper we will focus on the alternative architecture that is a Web-server cluster based on DNS dispatching mechanisms. This architecture does not present risks of IP-dispatcher bottleneck, and can easily scale from *locally* to *geographically* distributed Web-server clusters. The main problems of scheduling through the DNS are due to the high non-uniformity of the incoming load from different client domains [Arlitt and Williamson 1997; Cuhna *et al.* 1995], the high variability of real Web workload [Barford and Crovella 1998; Deng 1996], and WWW address caching mechanisms that let the DNS control only a very small fraction of the user requests. Although reducing to zero the period of address caching gives full control to the DNS, two facts hamper this solution for a Web cluster that receives heavy request load: the DNS would be subjected to become a bottleneck for scheduling and, most importantly, very small address caching periods (e.g., less than 120 seconds) are ignored at intermediate name servers in order to avoid overloading the network with name resolution traffic.

The limited control of the DNS poses a serious obstacle to load balancing. Previous performance studies have indicated that even small caching periods such as five minutes would reduce the DNS control to

a minor fraction of all requests reaching the Web-server cluster [Colajanni *et al.* 1998b]. These peculiarities make scheduling algorithms for traditional distributed systems inappropriate to DNS-based Web clusters and motivate the study of new assignment schemes that require additional system state information to control the load of the Web servers. In this paper, we start with studying dispatching policies that can perform well under theoretical or idealistic conditions that the DNS could have immediate access to any necessary state information. Then, we analyze these policies in light of their application to a DNS architecture working in a realistic environment. In particular, the role of past load information in DNS scheduling and state estimation heuristics are analyzed. It is worth to note that our interest goes only to the policies and estimators that can be actually applied in the WWW environment. Main requirements are low computational complexity, and compatibility with existing Web standards and protocols. Since we cannot force any WWW entity to collect load data for the DNS dispatcher, all information has to come from the Web cluster components, i.e., the cluster DNS and servers. As a result, we show that simple heuristics executed by the Web cluster components are able to provide all state information needed by the DNS scheduling policies, thereby demonstrating that the proposed scheduling algorithms can be used in an actual scenario. Incorporating these policies into a DNS leads to a Web-server cluster that provides much better performance than that achieved by DNS-based systems using scheduling policies based on unstable state information (e.g., least-loaded server) or no information at all (e.g., round-robin, random).

The paper is organized as follows. Section 2 points out the difficulties of DNS scheduling, and reviews the most promising policies focusing on the state information they require. Section 3 discusses whether and how this information can be obtained at the DNS. It examines several heuristics for estimating state information that can be integrated into the DNS scheduling policies. Section 4 presents the simulation model and parameters of the WWW components from the point of view of the Web-server cluster. Section 5 analyzes the performance of the proposed DNS policies for a large set of simulation scenarios. Section 6 concludes the paper with some final remarks.

2 DNS ALGORITHMS FOR LOAD CONTROL

The scalable Web-server cluster uses one URL-name to provide a single interface for viewers. The system consists of N homogeneous distributed servers that furnish the same set of documents, and a dedicated *cluster DNS* (CDNS) that translates the URL-name into the IP-address of one of the servers in the cluster. Besides the role of IP-address resolver, the CDNS performs as a dispatcher that distributes the requests among the servers based on some optimization criterion (e.g., load balancing, minimization of the system response time, minimization of overloading servers).

On the user side, the clients have a (set of) local name server(s) and are connected to the network through gateways. We will refer to the network sub-domain behind these local gateways as *domain*. From the point of view of the Web-server cluster, the client is only identifiable from its domain. There can also be many intermediate domain name servers spread out over the network between the path from a local name server to the CDNS.

A session is defined to be the entire period of access to the Web site from a single user, where many HTML page requests may be issued. Since an HTML page is typically composed of a collection of objects, each object request requires an access to the server. We will refer to them as *hits*. Any session of a client to the Web cluster consists of two phases: the *IP-address request* phase during which the client asks the CDNS for a translation of the Web cluster URL into the IP-address of one of the Web servers in the cluster, and the *Web document request* phase in which various pages are requested directly to the Web server selected by the CDNS. The IP-address request is initially submitted to the local name server of the client domain, because it typically caches the URL-name to IP-address mapping for a certain period, namely the *time to live* (TTL) interval, chosen by the CDNS. If the cache of the local name server has a valid mapping for this URL-name, the document request is sent directly to the Web server without requiring the CDNS to solve the IP-address request. Otherwise, the IP-address request is submitted to subsequent intermediate name servers, and only if the mapping is not cached in any of these, the IP-address request reaches the CDNS of the Web cluster. The CDNS returns the IP-address of one of the servers in the cluster and the TTL.

2.1 DNS scheduling issue

The non-uniform distribution of client requests among the domains and the high variability of real client load are the major problems that any dispatching policy (either IP-dispatcher based or DNS-based) has to address. However, there are other issues that make CDNS different from a normal scheduler and cause even major obstacles to the load balancing of the servers in a cluster. IP-address caching at local and intermediate name servers limits the control of the CDNS to a small fraction of the requests reaching the Web cluster. That is to say, during the TTL period bursts of requests can arrive from a domain to the same server, thereby causing hot spots [Cuhna *et al.* 1995], especially if the domain has many clients. These subsequent arrivals during the TTL are quite transparent to the CDNS. Moreover, the requirements below further constrain the potential alternatives for CDNS dispatching algorithms.

1. Low computational complexity because scheduling decisions are required in real-time.
2. Full compatibility with existing Web standards and protocols. Any assumption that requires modification of the WWW environment will not be pursued.

3. All state information needed by a policy has to be actually accessible on the CDNS. In particular, the CDNS and Web servers of the cluster are the only entities that can collect and exchange load information. Any state information that needs some active cooperation from any other WWW components, such as client browsers, intermediate name servers, and users, is not considered, because the previous requirement would be violated.

2.2 DNS dispatching algorithms

Scheduling policies such as round-robin and random used in [Kwan *et al.* 1995; Andresen *et al.* 1996; Mosedale *et al.* 1997] do not require any state information. However, it has been shown [Colajanni *et al.* 1998b] that these algorithms perform very poorly under realistic scenarios, e.g., when we consider the non-uniform distribution of the clients among the domains, and caching the URL-name to IP-address mapping for a TTL period greater than 0. In this section we review some strategies that can improve performance of DNS-based Web clusters under theoretical conditions that all state information are readily and instantly available. All better performing scheduling algorithms studied in the past tend to use additional state information in mapping URL names to IP-addresses. Hence, one important consideration in dealing with the scheduling problem is the kind of information that is actually available on the CDNS. Main results described in [Colajanni *et al.* 1998b; Colajanni *et al.* 1998a] which are of interest to our CDNS dispatching algorithms discussed in this paper can be outlined as follows:

- Even a frequent exchange of detailed information about the present and past load conditions of each Web server is not sufficient to provide scheduling decisions that can avoid overloading any server. The dynamics of the WWW, such as the high variability of domain and client workloads reaching a server, make the Web server load information obsolete quickly and poorly correlated with future load conditions. This excludes policies such as least-loaded server from further consideration.

- An effective scheduling policy has to take into account some domain information, because any CDNS decision on an IP-address request affects the selected server for the entire TTL interval during which the URL-name to IP-address mapping is cached in the name servers. Therefore, a centralized scheduler such as the CDNS needs to make an adequate prediction about the impact on the future load of the servers. The key goal is to estimate the *domain load rate* which is the average number of requests coming from a domain per second. We will discuss how it is possible to estimate this parameter rate in Section 3.

- The most useful information is a combination of domain and server information. Due to the high variability of the WWW scenario, the *domain load rate* may not always be a sufficient means to

estimate the actual load of the servers. Therefore, to be sure that the chosen server is not over-utilized, it is important to use a simple mechanism that monitors the actual load of each server and informs the CDNS when some server is over-utilized. In such a way, the CDNS can exclude it from any further assignment until its load falls below the threshold. (When all servers have sent alarm messages to CDNS, the CDNS will temporally stop making new assignment, e.g., by responding server unreachable.) We assume that all of following scheduling algorithms apply this feedback alarm mechanism.

Since the CDNS replies to an IP-address request through an (IP-address, TTL) pair, we partition the scheduling policies into two main classes: (1) algorithms with *constant TTL*, if the CDNS uses the same TTL for all requests; (2) algorithms with *adaptive TTL*, if the CDNS chooses dynamically the TTL most appropriate to an IP-address request. Here we analyze three algorithms with constant TTL and one with adaptive TTL.

Two-tier Round-Robin (RR2). This algorithm is based on two considerations [Colajanni *et al.* 1998b]. First of all, since the clients are not uniformly distributed, the load rate of each domain is typically very different. Secondly, the risk of overloading some of the servers is mainly due to the requests coming from a few very popular domains. Therefore, RR2 uses the domain load rate information to partition the domains connected to the Web cluster into two classes: *normal* domains and *hot* domains. In particular, RR2 sets a *class threshold* and evaluates the *relative domain load rate*, which is with respect to the total number of requests from all domains. The domains characterized by a relative load larger than the class threshold belong to the hot class. For default, we set the class threshold to $1/|domain|$, where $|domain|$ is the average number of domains connected to the servers. The RR2 strategy applies a round-robin policy to each class of domains separately. The objective is to reduce the probability that the hot domains are assigned too frequently to the same servers. Partitions of domains in more than two classes have been investigated with no performance improvement.

Dynamically Accumulated Load (DAL). A load control algorithm should take into account that any IP-address mapping done by the CDNS affects the selected server for the entire TTL interval during which the URL-name to IP-address mapping is cached. For this purpose, DAL [Colajanni *et al.* 1998b] obtains from the domain load rate an estimation of the so called *hidden load weight* which is the average number of requests that each domain sends to a Web server during a TTL interval after a new IP-address request has reached the CDNS. Each time the CDNS makes a server selection following an IP-address request, it accumulates the hidden load weight of the requesting domain in a bin for each server. At each new IP-address request, the CDNS selects the server that has the lowest bin level.

Minimum Residual Load (MRL). This algorithm is a modification of the basic DAL [Colajanni *et al.*

1998b]. The CDNS maintains an *assignment table* containing domain to server assignments and their times of occurrences. At the arrival of an IP-address request, the CDNS evaluates the expected number of *residual* requests that each server should have, on the basis of previous assignments, and chooses the server with the minimum number of residual requests. Different from the previous two policies that require only the domain load rate information, MRL needs also an estimation of the *mean session time* so as to evaluate the residual load.

Adaptive TTL (AdpTTL). This is an entirely different class of algorithms proposed in [Colajanni *et al.* 1998a]. It explores the TTL component that is transmitted by the CDNS to the client in reply of an IP-address request. The motivation for this approach comes from the observation that the number of page or hit requests following an IP-address request, independently of its origin domain, increases with the TTL value. However, a simple reduction of the TTL value with the purpose of giving more control to the CDNS does not work. Indeed, when intermediate name servers receive from a CDNS too low an expiration time value, they typically use their default TTL value for caching. Moreover, a generic reduction of the TTL does not help to balance the unevenly distributed client requests. The basic idea of AdpTTL is to assign a different TTL (lower or higher than the default value) to each client that submits an IP-address request. The value is tailored by taking into account the load rate of the domain that has originated the requests. In brief, the CDNS selects a server using one of the above constant TTL policies and assigns a low TTL when the address resolution requests come from more popular domains, and a high TTL when the requests are originated by small domains. In such a way, an *adaptive* TTL can reduce the skews on subsequent Web page or hit requests following an IP-address requests.

The AdpTTL algorithm considered in this paper uses a RR2 policy to choose a Web server, and then selects the appropriate TTL value based on each domain load rate. Specifically, the assigned TTL value is inversely proportional to the load rate of the domain from which the IP-address request has been issued.

3 HEURISTICS FOR ESTIMATING STATE INFORMATION

3.1 Gathering information at the DNS

All the CDNS policies in the previous section base their decision on the estimated domain load rates, and the feedback alarms coming from the overloaded Web servers. Since we cannot force any modification

on the WWW components external to the cluster to transmit some information to the DNS, any data that keeps trace of load weights has to be collected on the CDNS by the entities of the Web cluster, i.e. the CDNS itself or the Web servers.

The information coming directly from the clients to the CDNS is very limited because the CDNS sees only the IP-address request of a client's domain after a TTL period. For performing URL-name to IP-address mapping of a non-clustered (or non-mirrored) Web site, this information is sufficient. However, this makes it very difficult to take dispatching decisions in a Web cluster. In particular, the CDNS cannot estimate the domain load rate by collecting the number of IP-address requests originated by each domain, because due to address caching mechanisms, the CDNS will see another IP-address request coming from the same domain only after the TTL period, independently of the domain load rate. Hence, we can expect that, without additional state information to the CDNS, the cluster would not perform well due to the poor quality of the domain load rate estimation.

The only viable approach requires an active cooperation of the Web servers with the CDNS. The servers can track and collect all load information that represents the workload to the Web cluster based on the domain that has originated it. This technique to estimate the domain load rate requires an exchange of periodic messages from the Web servers to the CDNS, while the feedback alarm mechanism is based on asynchronous messages.

The implementation of the *feedback alarm* information requires two simple mechanisms: a load monitor/checker on each Web server, and an asynchronous communication mechanism between each server and the CDNS. Each server periodically calculates its utilization (the period can be of 8 or 16 seconds) and checks whether it has exceeded the load threshold (typically set to 15–20% above the average cluster utilization). In that case, the server sends an *alarm signal* to the CDNS to exclude it from the table of available servers until its load falls below the load threshold. This last event is communicated to the CDNS through a *normal signal*. If every server in the cluster generates an alarm and the table is empty, the CDNS replies to IP-address requests that the server is unreachable or too busy to respond. However, this is rare event and for all proposed policies did not occur more than very few times in our experiments.

The Web servers can estimate the domain load rate through the *logfile* maintained by each server to trace the client accesses. Each server periodically sends its partial view estimate of the domain load rates to the CDNS, where a *load collector* process gathers all estimates and computes the global domain load rates.

Figure 1 summarizes the additional software components (grey in the figure) needed for an efficient DNS-based Web cluster. In addition to the DNS base function, the system includes a dispatcher that works as server and TTL selector, an alarm monitor, and a load collector. The server selector assigns each address request to one of the Web server based on some dispatching algorithm discussed in Section 3.2. The

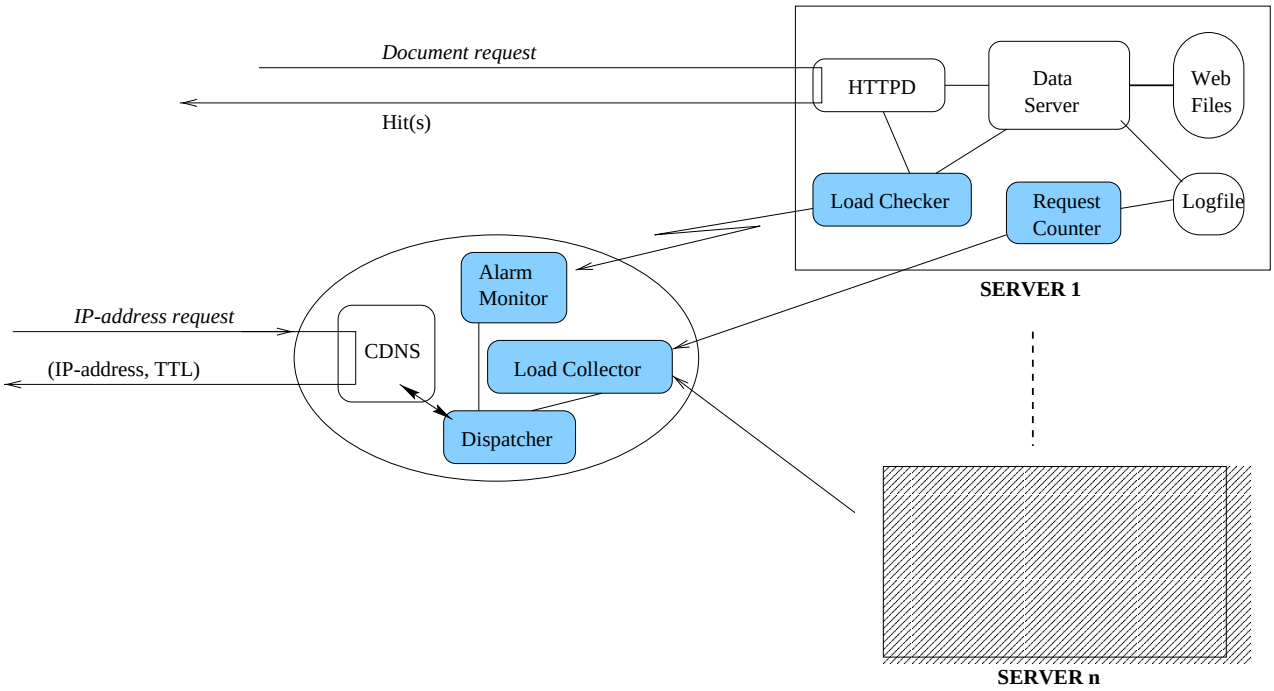

Figure 1: Architecture of the DNS-based Web cluster.

TTL selector sets the appropriate TTL value for the adaptive TTL algorithms. The alarm monitor tracks the feedback alarm from servers to avoid assigning requests to an overloaded server until the load level is returned to normal. The load collector gathers the domain load information from each server and estimates the domain load rate (and, when necessary, the hidden load weight) of each connected domain.

Also shown is the corresponding set of components in one Web server of the cluster. Besides the HTTP daemon server, the proposed CDNS algorithms require a load checker and a request counter. The load checker tracks the server utilization and issues alarm and normal signals accordingly as explained above. The request counter estimates the number of requests received from each domain in a given period through the analysis of the *logfile*, and periodically provides the information to the load collector in CDNS.

When the load collector does not receive any message from the request counter of a Web server after a certain period, it assumes that the server is unreachable, faulty or overloaded. In such a case, the dispatcher excludes this server from the table of available servers as though an alarm signal was received by the alarm monitor.

3.2 Information for domain load rate estimation

The domain load rate can be estimated through a function of the total number of hits issued to a

server from a domain D in a given period of observation and then normalized to one unit of time, i.e.,

$$W_{sec}^{D} = \sum_{i=1}^{S^D} \sum_{j=1}^{R_i^D} h_{i,j}^{D} \tag{1}$$

where S^D is the number of sessions, R_i^D is the number of page requests issued to a server during the $i-$th session (S_i^D), and $h_{i,j}^D$ is the number of hits in the j-th page requested during the $i-$th session (S_i^D).

The load information can be collected by the Web servers based on different granularity of details: from the number of sessions to the number of page requests to the number of hits. These approaches are referred to as *WS.ses, WS.req,* and *WS.hit*, respectively. Let us analyze in more details how it is possible to obtain this information. We note that the *logfile* maintained by each server to trace the client requests contains the accesses in terms of hits. According to the Common Logfile Format [W3C 1995], the information reported for each hit includes the remote (domain) hostname (or IP-address), the requested URL, the date and time of the request and the request type. Furthermore, there are also extended logs to provide referred information for linking each request to a previous Web page request from the same client.

Sessions. The mean number of sessions from each domain is a rough approximation of the domain load rate, namely *domain session rate.* This view of Equation (1) assumes that the average user behavior is similar in terms of the average number of pages requested in a session and the average number of hits in a page. Moreover, getting this session information may not be as straightforward as one might think. Since a typical HTTP request is pseudo-anonymous in the sense that the server knows only the IP-address or name of client's domain, two requests from the same domain are likely to come from the same client but certainly do not have to. A session can be identified via a *cookie generation* mechanism or inferred through some heuristics. Through a non-persistent cookie, it is possible to determine the beginning of each session from a new client, while the end of a session can be determined through a session limit [Kristol and Montulli 1997]. Heuristics for identifying users without cookies can use the site's topology or referrer information. For example, the algorithm proposed in [Pirolli *et al.* 1996] checks if a requested page can be directly reached from the set of already visited pages. Session information is of more interest to the MRL policy that needs to estimate the mean session length in addition to the domain load rate.

Page requests. The mean number of page requests from each domain is an alternative approximation of the domain load rate, namely *domain page rate.* This value can be measured for each domain by excluding non-HTML requests (e.g., gif files) from the counting. Implicitly, it is assumed that the average number of hits per page request is similar for all domains. Since the server can track the total number of hits and the total number of page requests, the evaluation of the mean number of hits $\overline{h_i}$ per page request

is straightforward. Once known this mean, from (1) the CDNS can evaluate the domain page rate as $\sum_{i=1}^{|S^D|} R_i^D \overline{h_i}$.

Hits. As shown in Equation (1), estimating the workload as a function of the number of hits from each domain is the most accurate way to obtain the domain load rate, namely *domain hit rate*. Another advantage is that this information is the less expensive to get as it requires only a sequential scan of the logfile. In such a way, each entry in this file coming from a domain is counted as an hit for that domain. Moreover, we can also consider the hit dimension measured in bytes.

However, even reporting of hits as a measure of load causes some approximation. The noise is due to the fact that the client access to the Web pages can follow different paths through the hyperlink structure. This causes some variability because only the first reference to an object from a client requires an access to the Web server, while the successive requests for the same object are found in the client cache. Therefore, the hit request rates from a domain may vary, even if the same set of pages are requested from a domain during a certain period [Pitkow 1997].

3.3 Heuristic analysis

We focus on heuristics to estimate the domain load rate that can be usefully applied in a real-time environment. In particular, they must have low computational complexity, and require state information accessible to the CDNS. For example, although the accuracy of the *time series method* [Box *et al.* 1994] is superior to other approaches when the system is subject to non-stationary load behavior, its computational complexity and the very large sample size requirement make this method not suitable to a system that requires real-time decisions.

The most practical approach is to use all data collected during the last *sampling period*. Past load data is commonly used to forecast future load and improve scheduling performance. The first issue is the specification of the sampling period. After that, we can use two heuristics:

Simple mean. All information collected in the sampling period are given equal importance. The data transmitted to the CDNS is a simple mean of all observations done in the sampling period.

Weighted mean. The sampling period is partitioned into K intervals. The observed value of each interval is combined with a different weight (higher for more recent observations) obtained by a decay distribution. The estimated load rate for a domain D in a second is given by

$$w^D = \left(1/\sum_{i=1}^{K} e^{-i/2}\right) \sum_{j=1}^{K} e^{-j/2} r^D(j) \qquad (2)$$

where $r^D(j)$ for $j \in \{1, \ldots, K\}$ are the requests from the domain D observed in the j-th interval. Following the results of the previous section, the observed requests can be sessions, pages or hits.

The first set of simulation experiments aimed at analyzing how well the described heuristics estimate the relative domain load rates in a static environment in which the connected domains maintain the same request rates.

The experiments confirmed that using the CDNS information alone (i.e., a subset of the sessions to the Web cluster, evaluated through the number of IP-address requests) is not able to capture the skew of the relative domain load rates, because in a given period the CDNS sees the same number of IP-address requests coming from each domain. Hence, no further study is presented on the performance of scheduling policies based on CDNS information only.

On the other hand, any evaluation carried out through Web server information, such as number of sessions, requests and hits, can provide an acceptable estimation of the relative domain load rate, over a rather long period of observation, for any kind of distribution of the clients among the domains.

Another interesting result is that, at least for the estimation of the relative domain load rate over a long period, there is not much difference in using a simple mean, a weighted mean on two intervals, and a weighted mean on four intervals. Although most of the experiments were carried out for a sampling period of 480 seconds, the results were almost insensitive to the value of the sampling period. Indeed, similar results were obtained for different sampling periods such as 240, 360 and 600 seconds, while worse results are obtained only for much shorter periods such as 90 seconds or less.

4 SIMULATION MODEL

4.1 Model assumptions and parameters

We assume that clients are partitioned among the domains based on a Zipf's distribution, i.e., a distribution where the probability of selecting the i-th ranked domain is proportional to $1/i^{(1-x)}$ [Zipf 1949]. Generally speaking, if one ranks the popularity of domains by the frequency of their accesses to the Web server, the distribution on the number of clients in each domain is a function with a short head (corresponding to big providers, organizations and companies, possibly behind firewalls), and a very long tail. For example, a workload analysis on academic and commercial Web sites shows that in average 75% of the client requests come from only 10% of the domains [Arlitt and Williamson 1997]. In the experiments, the clients are partitioned among the domains based on a Zipf's distribution with parameter $x = 0.2$.

Since the focus of this paper is on Web cluster performance, we did not model the Internet traffic [Brau and Claffy 1995], but we consider major components that impact the performance of the Web-server cluster. For example, we included the *intermediate name servers* that typically exist between a domain and the Web-server cluster. The model consider these name servers for their impact on operations and performance of the CDNS scheduling algorithms. The mean number of intermediate servers that an address resolution request has to cross from its local name server to the CDNS is chosen equal to the mean number of hops shown in [Bestavros 1997]. The function is similar to a double bell Gaussian distribution with peaks around 2 and 16 corresponding to local and remote requests, respectively.

We consider all the details concerning a client *session*. In particular, we model both the *IP-address request* phase during which the client asks the CDNS for a translation of the Web cluster URL into the IP-address of one of the Web servers in the cluster, and the *Web document request* phase in which various pages are requested directly to the Web server selected by the CDNS. When the CDNS returns the IP-address of one of the servers in the cluster and the TTL, our model assumes that each intermediate name server along the path from the CDNS to client's domain caches this mapping for the TTL period.

The model incorporates all main characteristics of real Web workload, and in particular the self-similar nature of Web traffic [Crovella and Bestavros 1997]. The high variability in the workload is represented through a heavy-tailed distribution, which is a distribution where $P[X > x] \sim x^{-\alpha}$ as $x \to \infty$ for $0 < \alpha < 2$.

Typical heavy-tailed distributions used for representing the Web workloads are the Pareto and Weibull distributions [Barford and Crovella 1998; Crovella and Bestavros 1997; Deng 1996]. The probability density function of the Pareto distribution is given by $p(x) = \alpha k^{\alpha} x^{-\alpha-1}$, where $\alpha,\ k > 0$ and $x \geq k$. Its distribution function is given by $F(x) = P[X \leq x] = 1 - (k/x)^{\alpha}$. The probability density function of the Weibull distribution is given by $p(x) = \frac{a x^{a-1}}{b^a} e^{-(x/b)^a}$ [Law and Kelton 1991].

The number of *page requests* per session is assumed to be exponentially distributed. The time between retrieval of two successive Web pages from the same client, that is the *user think time*, is modeled through a Pareto distribution [Barford and Crovella 1998; Deng 1996]. The number of *embedded references* per page, that is the number of hits excluding the main HTML page, is also obtained from a Pareto distribution [Barford and Crovella 1998; Mah 1997].

The *inter-arrival time of hit requests* to the servers, that is the processing time spent by the browser parsing Web files and preparing the new TCP connection, is assumed to be heavy-tailed Weibull distributed [Barford and Crovella 1998]. The *hit size distribution*, which is the distribution of the files requested to a Web server (this index differs from the distribution of the file sizes in the server's file system modeled through a Lognormal and a Pareto), is obtained from a heavy-tailed Pareto distribution [Barford and Crovella 1998]. This distribution is also taken as a measure of the Web server capacity denoted in served bytes per

Category	*Parameter*	*Value (default)*
Web cluster	Number of servers	7
	Server capacity	$5 \cdot 10^{-7}$ second per byte
	Average cluster utilization	0.6667
Domain	Connected	50
	Client distribution among domains	Zipf ($x = 0.2$)
	TTL (constant)	240
	TTL (adaptive)	[60–3000]
	Inter-arrival time of new domains	1000
	Inter-rerank time of domains	[100-1000] (250)
Client	Number	2500
	Web page requests per session	exponential (mean 10)
	User think time	Pareto ($\alpha = 1.5$, $k = 3$)
	Embedded references per Web page	Pareto ($\alpha = 2.43$, $k = 2.3$)
	Inter-arrival time of hits	Weibull ($a = 0.382$, $b = 0.146$)
	Hit size request	Pareto ($\alpha = 1.25$, $k = 1800$)

Table 1: Parameters of the system (all time values are in seconds).

second. The cluster average utilization is always kept to 66% of the whole capacity in all experiments. This value is obtained as a ratio between *system load*, that is the average number of bytes per second requested from the Web cluster, and the *cluster capacity* which is the sum of the capacities of all Web servers in the server cluster.

The *constant TTL* algorithms use a TTL set to 240 seconds and the *adaptive TTL* policy uses TTL ranging from 60 seconds for the hottest domain to higher values (reaching even 3000 seconds) for the other domains, while maintaining a similar overall request rate to the CDNS. Other parameters for the CDNS scheduling algorithms are set on the basis of the results achieved in [Colajanni *et al.* 1998b], because a detailed search for their optimal choice is out of the scope of this paper.

A summary of the model distributions and other parameters that we used in our simulations are reported in Table 1.

4.2 Performance evaluation criterion

The main goal of this study is to investigate the impact of the CDNS dispatching algorithms on

avoiding that some server becomes overloaded, while others are underutilized. Since the load balance of the overall Web server cluster is only an indirect goal, popular metrics such as the standard deviation of server utilizations are not useful for our purposes. Therefore, we define the *cluster maximum utilization* at a given instant as the highest server utilization at that instant among all servers in the cluster. For example, assume three servers in a Web-server cluster subject to an offered load that in average is about 66% of the overall system capacity. If the server utilizations are 60%, 75% and 63%, respectively, at time t_1, and 93%, 66% and 42%, respectively, at time t_2, the cluster maximum utilization at t_1 is 75% and that at t_2 is 93%. With a cluster maximum utilization of 93%, the Web-server cluster has some load balancing problem at t_2, because the average offered load is about 66%.

Specifically, the major performance criterion is the *cumulative frequency* of the cluster maximum utilization, i.e., the probability or fraction of time that the cluster maximum utilization is below a certain value. By focusing on the highest utilization among all Web servers, we can deduce whether the Web cluster is overloaded or not. Hence, the performance of the various scheduling policies is evaluated by tracking at periodic intervals the cluster maximum utilizations observed during the simulation runs. The server with the maximum utilization changes over time. However, if the cluster maximum utilization at an instant is low, it means that no server is overloaded at that time. By tracking the period of time the cluster maximum utilization is above or below a certain threshold, we can get an indication of how well the Web-server cluster is running.

5 PERFORMANCE RESULTS

In this paper, we focus on the CDNS dispatching policies that were demonstrated to perform well under theoretical scenarios with ideal state estimators (where the variability of Web workloads was assumed to be exponentially distributed) [Colajanni *et al.* 1998b; Colajanni *et al.* 1998a]. In particular, a *theoretical* algorithm is assumed to have immediate access to any information which is needed to obtain the precise server load status and the exact domain load rates. However, it should be noted that even with a detailed and exact knowledge of the current load condition, this algorithm does not necessarily perform optimally under real scenarios because of the high variability of the load conditions. Other unexpected irregularities can be due to the address caching mechanisms.

We compare the performance of the dispatching policies combined with some heuristics for state estimate against the theoretical versions of the same algorithms for different load conditions under two scenarios: *static* (i.e., the average request rate from each connected domain does not change over time), and *dynamic* (i.e., the domain load rates are subject to dynamic variations to be explained in Section 5.2).

5.1 Static scenario

A *static scenario* is an environment in which the client requests arrive from a given set of domains, where each domain is with a given request rate during the entire period of the simulation study. Each simulation run is made up of three hours of the Web site activities. Confidence intervals were estimated, and the 95% confidence interval was observed to be within 4% of the mean.

In the figures, *WS.ses*, *WS.req* and *WS.hit* refer to the estimation carried out using various Web server information that is, number of sessions, requests and hits, respectively.

Figures 2–5 show the performance of the four policies (RR2, DAL, MRL, and AdpTTL), where the estimation of the domain load rate is done through the simple mean approach on a sampling period of 240 seconds. Also shown is the round-robin (RR) scheme for comparison purpose. In these figures, the x-axis is the cluster maximum utilization of the Web server cluster, and the y-axis is the cumulative distribution. For example, Figure 2 shows that for round-robin, $Prob(MaxUtilization < 0.95) \approx 0.50$. This means that under RR at least one Web server is overloaded (i.e., utilized above 0.95) for half of the time.

These figures demonstrate that in a static scenario any one of the heuristic estimates works very well under RR2, MRL and AdpTTL policies, and there is not much difference among the different types of observed load information for estimating the domain load rate. Comparing the performance under the different heuristic estimation algorithms, we see that the final results do not always reflect the expectation that the number of hits should approximate the domain load rate better than the number of requests, and this latter should be better than the number of sessions. This order is clearly valid only for DAL policy, for which the number of sessions is a very poor load information. The performance of the other scheduling policies using the heuristics for state estimate is quite close to that obtained by the theoretical algorithm and sometime even better. In particular, the estimate based on hits provides the best performance under DAL policy (Figure 3). For the RR2 policy, the heuristic estimate based on sessions gives the best results (Figure 2), while for the MRL and AdpTTL policies there is no appreciable difference among the three kinds of load estimates (Figure 4 and 5, respectively). The similar results obtained for heuristics based on hits and requests were actually expected because both of them are a good source of information for estimating the domain load rate. The good results relative to the sessions for the RR2 are mainly due to the fact that this policy uses the *relative* domain load rate for a simple goal of partitioning the domains into two classes.

Conventional policy (such as RR) which does not consider any state information performs very poorly. A limited state information, such as that used by RR2, helps this policy to improve the performance with respect to RR (Figure 2). The probability of having no overloaded server improves from 0.5 to 0.8. However, better results are achieved by DAL, MRL and mainly AdpTTL. The probability that some server is over-

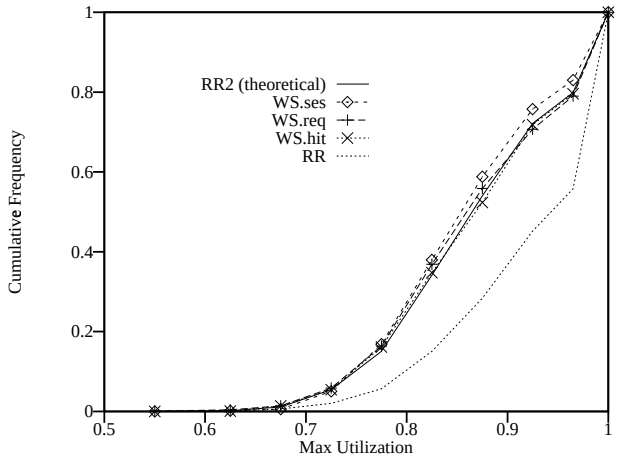

Figure 2: Performance of RR2 policy (*static scenario*).

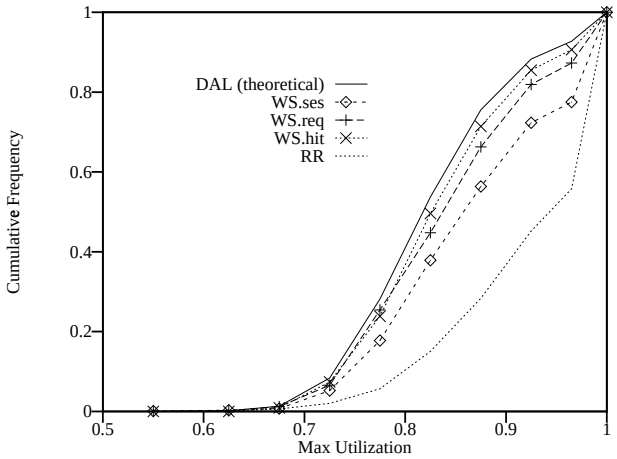

Figure 3: Performance of DAL policy (*static scenario*).

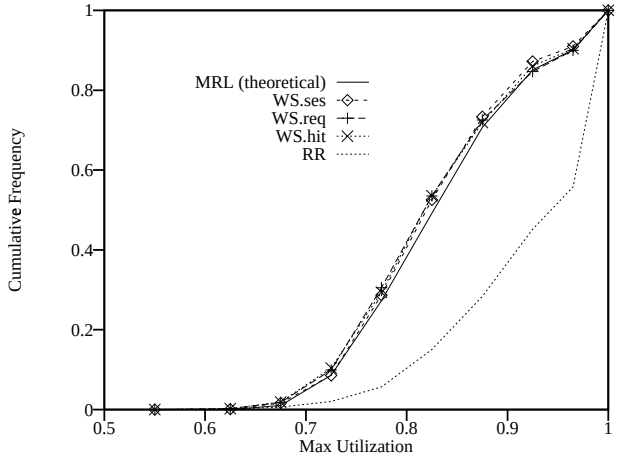

Figure 4: Performance of MRL policy (*static scenario*).

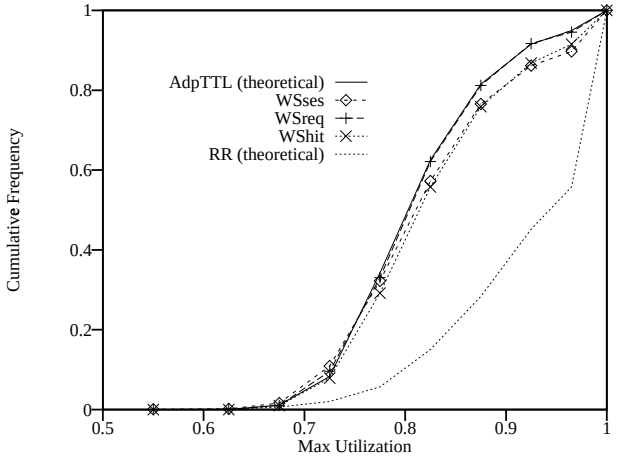

Figure 5: Performance of AdpTTL policy (*static scenario*).

loaded (i.e., it has a utilization higher than 0.95) is almost always below 0.1 (see Figures 3–5). However, it is the AdpTTL policy with adaptive TTL that achieves the best performance. Figure 5 shows that a CDNS using this scheduling algorithm almost always (i.e., with probability close to 0.95) maintains a utilization lower than 0.92 for all servers.

5.2 Dynamic scenario

Let us now proceed to consider the *dynamic scenario* which makes it more difficult for the heuristics to obtain accurate estimates of the client and domain load rate. In particular, we consider two events that modify the type of connections to the Web cluster:

Internal change. The connected domains remain the same, but there is a reclassification of their relative rank on request rate. (Recall that under the Zipf distribution, request rate from the i-th ranked domain is proportional to $1/i^{(1-x)}$ [Zipf 1949].) In other words, we assume a modification of the load rate of some domain.

External change. Some domains leave the Web cluster, while connections from new domains arrive. In particular, the domains with the ten lowest ranks are replaced by new domains that can reach a random position between the 11-th and the last one. These external changes are realistic because it is likely that a total disconnection occurs from a domain having less active connections to the Web-server cluster, and it is quite unlikely that connections from new domains suddenly reach the top 10 ranks.

To achieve a 95% confidence interval within 4% of the mean, each simulation run is now made up of five hours of the Web site activities.

Figures 6–9 refer to a dynamic scenario in which the *inter-rerank time* of already connected domains is set to 250 seconds, and the *inter-arrival time* of new domains is set to 1000 seconds. As expected, all policies perform worse than in the static scenario. However, the results of the proposed algorithms still resemble those obtained by the theoretical versions, and are much better than the results achieved by RR.

Comparing the performance under the different heuristic estimation algorithms, we see that now the number of hits approximates the domain load rate better than the number of requests and sessions. In particular, the estimate based on hits provides the best performance for MRL and AdpTTL policies (Figure 8 and 9, respectively). It also provides results that are very close to the best for RR2 and DAL policies (Figure 6 and 7, respectively).

We can summarize that RR2 and DAL are the policies that have the closest results to their theoretical version. However, AdpTTL is still the policy that performs the best even if its results are not as close to the theoretical version of this algorithm.

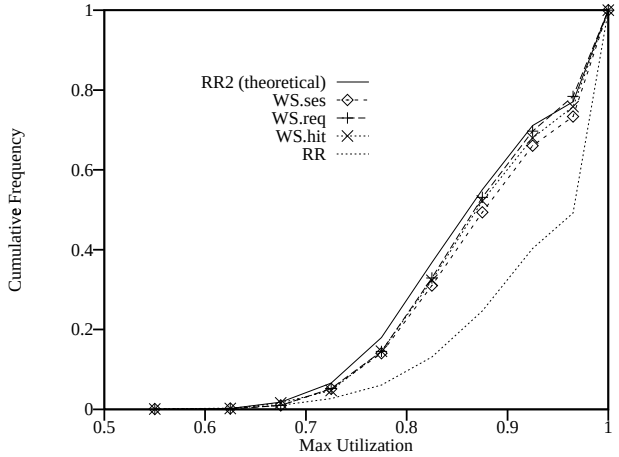

Figure 6: Performance of RR2 policy (*dynamic scenario*).

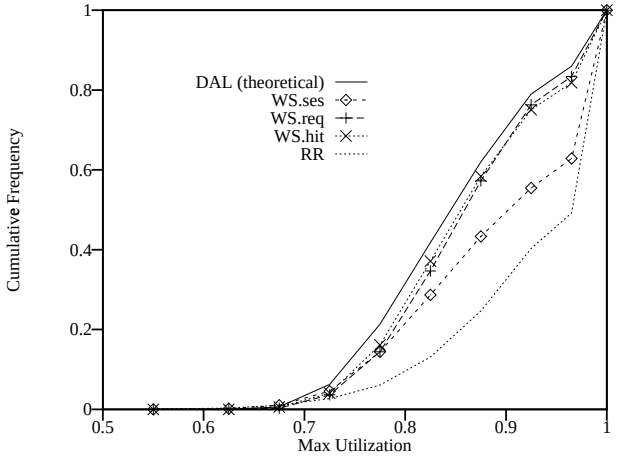

Figure 7: Performance of DAL policy (*dynamic scenario*).

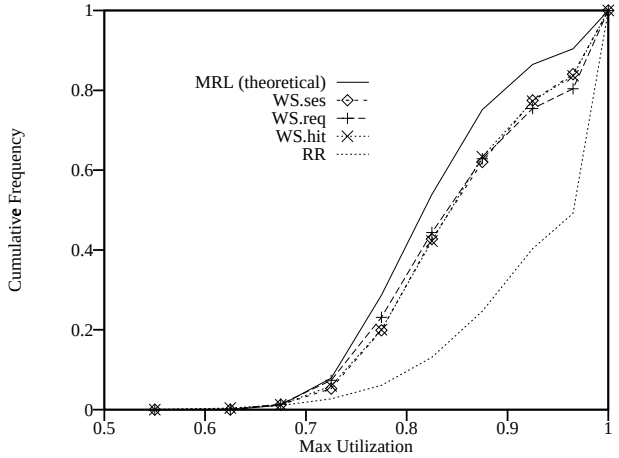

Figure 8: Performance of MRL policy (*dynamic scenario*).

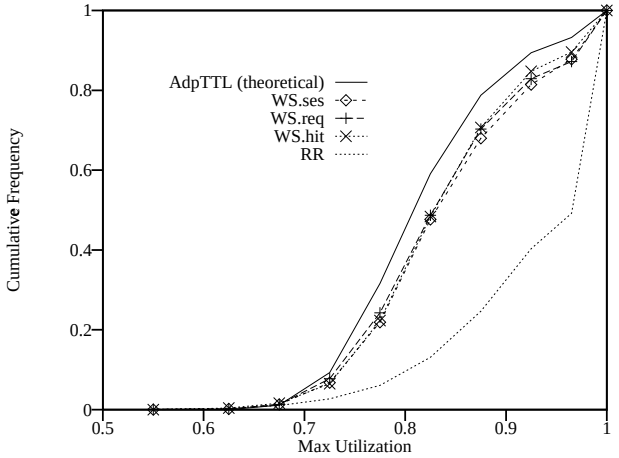

Figure 9: Performance of AdpTTL policy (*dynamic scenario*).

5.3 Sensitivity analysis

In the last set of experiments we evaluate the sensitivity of the scheduling policies and related heuristic to various system parameters. For these results, a different performance metric is used where we show the 96 percentile of the cluster maximum utilization, i.e., the $Prob(MaxUtilization < 0.96)$. In other words, the probability that no server of the Web cluster is overloaded (or exceeding 96% utilization) is considered as the metric.

First of all, we evaluate the sensitivity to the *inter-rerank time* under different sampling periods. The load information used for Figures 10–13 is based on *WS.hit*. In these figures, the x-axis goes from a very dynamic scenario (of 100 second inter-rerank time) to a more static scenario (of 1000 second inter-rerank time).

The Figures 10–11 show that all dispatching policies but RR2 are quite robust (i.e. the probabilities of a not overloaded cluster are never below 0.8) and in most cases (when the inter-rerank time is above 250 seconds) they are almost insensitive to the degree of variability of the scenario on inter-rerank time as they do not improve much as the variability diminishes. In particular, the AdpTTL with short sampling periods is the policy less sensitive to the variability of the scenario on inter-rerank time.

The sensitivity of the dispatching policies to the sampling period can be observed through a cross comparison of Figures 10–12 . When the variability of the scenario is low, i.e., the inter-rerank time is above 800 seconds, there are no appreciable differences among the length of the sampling period. On the other hand, for highly dynamic scenarios, short sampling periods are preferable. This agrees with the intuition that high variability makes past load information poorly correlated with future load.

For the purpose of fairly comparing the *simple mean* to the *weighted mean* approach, we consider two figures with same sampling period equal to 480 seconds: Figure 12 applies the simple mean to the entire sampling interval; Figure 13 partitions each sampling period into four sub-intervals of 120 seconds and applies the decay distribution in Equation 2. A comparison between these figures shows the usefulness of the decay distribution approach when the sampling period is relatively large (480 seconds or more). Applying a weight on the collected state data tends to give more stable result, especially when the CDNS uses the MRL or AdpTTL policy.

We now examine the sensitivity of the results to the distribution of client requests among the domains. As discussed in Section 4.1, the clients are typically partitioned among the domains based on a Zipf's distribution. Figures 14 and 15 show the performance of the proposed policies as a function of the Zipf parameter on the x-axis for a static and dynamic scenario, respectively. The Zipf's function goes from the highest skewed partition (i.e., with parameter $x = 0$) to the uniform distribution (i.e., with parameter $x = 1$).

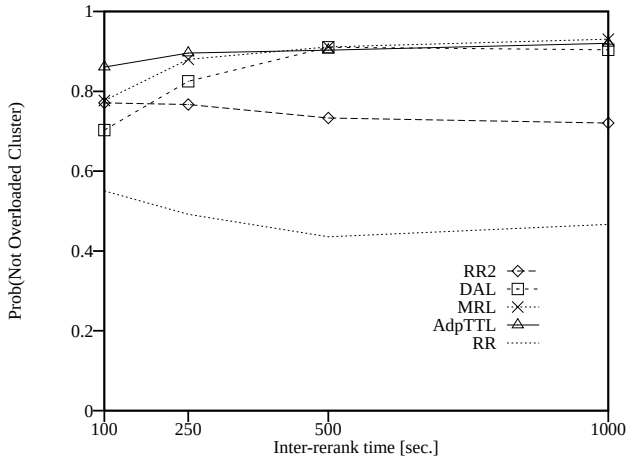

Figure 10: Sensitivity to the inter-rerank time of the domains using a simple mean on one sampling period of 120 seconds.

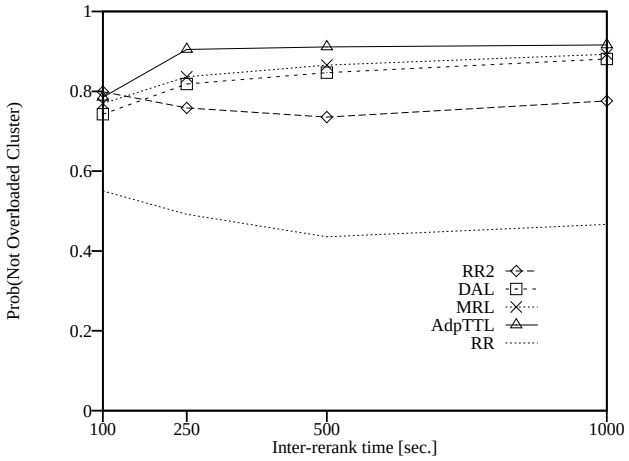

Figure 11: Sensitivity to the inter-rerank time of the domains using a simple mean on one sampling period of 240 seconds.

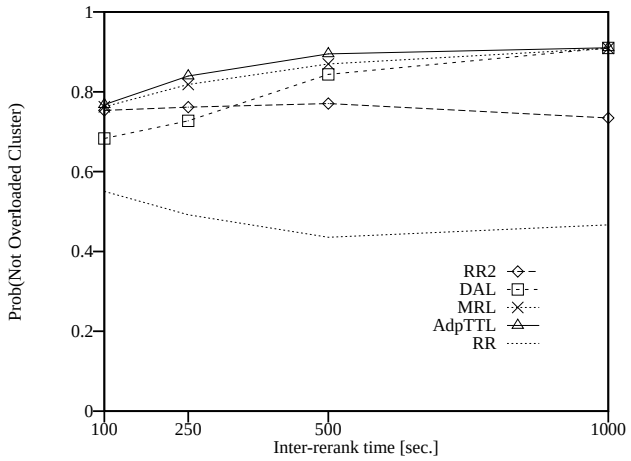

Figure 12: Sensitivity to the inter-rerank time of the domains using a simple mean on one sampling period of 480 seconds.

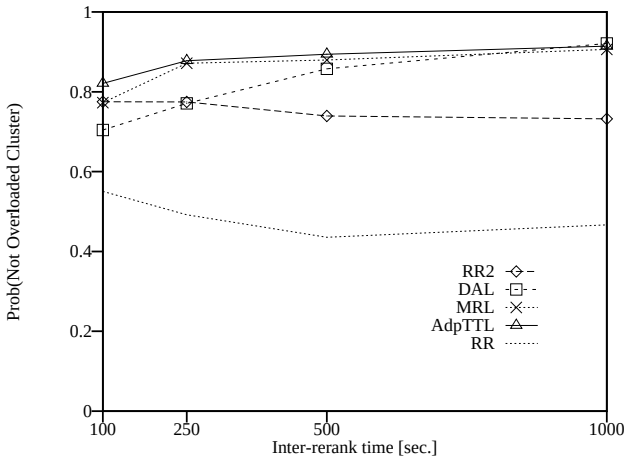

Figure 13: Sensitivity to the inter-rerank time of the domains using the decay distribution on four sampling periods of 120 seconds each.

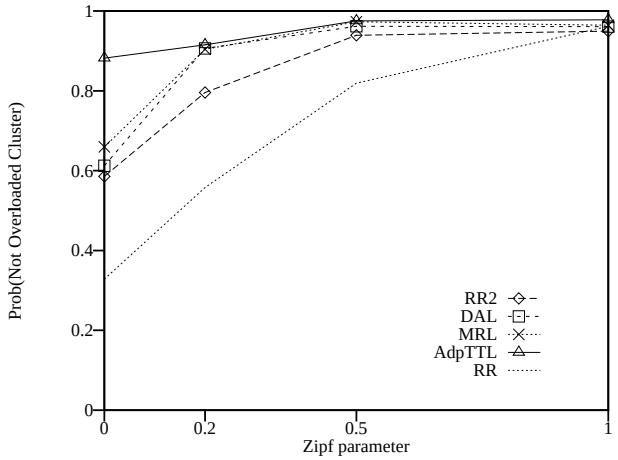

Figure 14: Sensitivity to the distribution of client requests among the domains (*static scenario*).

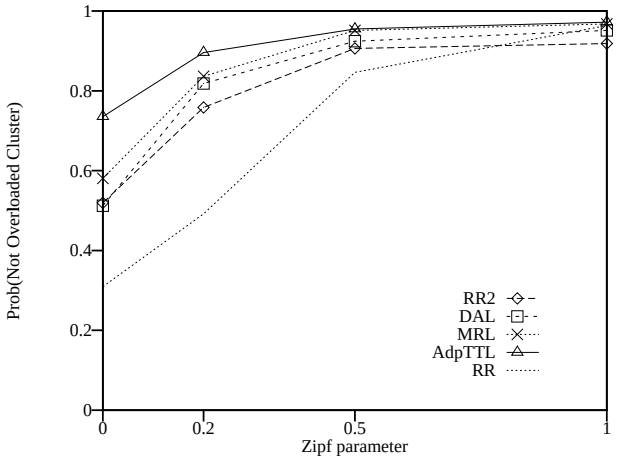

Figure 15: Sensitivity to the distribution of client requests among the domains (*dynamic scenario*).

AdpTTL remains the best policy for both static and dynamic scenarios. This is especially when the client distribution differs from the uniform distribution. Except for the case $x = 0$, under AdpTTL there is no server overloaded for more than 90% of the time in Figures 14 and 15. When the skew of the distribution decreases, all proposed policies show good performance. As expected, even the RR works fine when the clients are uniformly distributed among the domains. However, this assumption would be highly unrealistic.

6 CONCLUSIONS

Replication of information among multiple Web servers is necessary to support high request rates to popular Web sites. A clustered Web server organization is preferable to multiple independent mirrored-servers. Indeed, a Web cluster maintains a single interface to the viewers and has the potential to be more scalable, fault-tolerant and better load balanced. In this paper we have studied a Web-server cluster architecture in which the cluster DNS (CDNS) also plays the role of a cluster scheduler.

This paper considers DNS dispatching policies that perform well under theoretical conditions where all state information is readily available and the Web workload is typically exponentially distributed. It extends them to realistic Web scenarios characterized by long tail distributions for modeling the load, and assumes that state information needs to be estimated with low computation and communication overhead so as to be applicable to actual CDNS. We analyze the feasible sources and types of information that can be used to estimate the state information needed for each policy.

Finally, we conduct simulations to show that by incorporating these polices into the CDNS, the Web

cluster can achieve much better performance than other naive policies not using state information, such as round-robin and random, or relying on highly unstable information, such as least-loaded server. Several experiments demonstrate that the scheduling policies integrated with some heuristics for load estimation are still effective, even in the presence of highly skewed load in both static and dynamic scenarios. In particular, the adaptive TTL algorithm gives the best results even if it is the most sensitive to the quality of the heuristics.

Acknowledgments

We thank the anonymous referees for their constructive comments and suggestions, which helped to improve the quality of the final paper. The first two authors acknowledge the support of the Ministry of University and Scientific Research (MURST) in the framework of the project "Design Methodologies and Tools of High Performance Systems for Distributed Applications".

REFERENCES

Anderson, E., D. Patterson, and E. Brewer (1996), "The Magicrouter, an application of fast packet interposing," http://www.cs.berkeley.edu/~eanders/projects/magicrouter/osdi96-mr-submission.ps.

Andresen, D., T. Yang, V. Holmedahl, and O. H. Ibarra (1996), "SWEB: Toward a scalable World Wide Web server on multicomputers," In *Proceedings of the 10th International Symposium on Parallel Processing (IPPS'96)*, Honolulu, HI, pp. 850–856.

Arlitt, M. F. and C. L. Williamson (1997), "Web server workload characterization: The search for invariants," *IEEE/ACM Transactions on Networking 5*, 5, 631–645.

Barford, P. and M. Crovella (1998), "Generating representative Web workloads for network and server performance evaluation," In *Proceedings of Performance '98/ACM Sigmetrics '98*, ACM Press, Madison, WI, pp. 151–160.

Bestavros, A. (1997), "WWW traffic reduction and load balancing through server-based caching," *IEEE Concurrency 5*, 1, 56–67.

Box, G. E. P., G. M. Jenkins, G. C. Reinsel, and G. Jenkins (1994), *Time Series Analysis: Forecasting and Control*, Third Edition, Prentice Hall.

Brau, H. W. and K. C. Claffy (1995), "Web traffic characterization: An assessment of the impact of caching documents from NCSA's Web server," *Computer Networks and ISDN Systems 28*, 37–51.

Cisco (1997a), "Cisco's DistributedDirector," http://www.cisco.com/warp/public/751/distdir/.

Cisco (1997b), "Cisco's LocalDirector," http://www.cisco.com/warp/public/751/lodir/.

Colajanni, M., P. S. Yu, and V. Cardellini (1998a), "Dynamic load balancing in geographically distributed heterogeneous Web-servers," In *Proceedings of the IEEE 18th International Conference on Distributed Computing Systems (ICDCS'98)*, IEEE Computer Society Press, Amsterdam, The Netherlands, pp. 295–302.

Colajanni, M., P. S. Yu, and D. M. Dias (1998b), "Analysis of task assignment policies in scalable distributed Web-server system," *IEEE Transaction on Parallel and Distributed Systems 9*, 6, 585–600.

Crovella, M. and A. Bestavros (1997), "Self-similarity in World Wide Web traffic: Evidence and possible causes," *IEEE/ACM Transaction on Networking 5*, 6, 835–846.

Cuhna, C., A. Bestavros, and M. Crovella (1995), "Characteristics of WWW client-based traces," Technical Report BUCS-95-010, Department of Computer Science, Boston University, Boston, MA.

Deng, S. (1996), "Empirical model of WWW document arrivals at access link," In *Proceedings of the IEEE International Conference on Communication (ICC'96)*, IEEE Computer Society Press, Dallas, TX, pp. 1797–1802.

Dias, D. M., W. Kish, R. Mukherjee, and R. Tewari (1996), "A scalable and highly available Web server," In *Proceedings of the 41st IEEE Computer Society International Conference (COMPCON'96)*, IEEE Computer Society Press, pp. 85–92.

Kristol, D. M. and L. Montulli (1997), "HTTP State Management Mechanism (Rev1)," ftp://ds.internic.net/internet-drafts/draft-ietf-http-state-man-mec-04.ps.

Kwan, T. T., R. E. McGrath, and D. A. Reed (1995), "NCSA's World Wide Web server: design and performance," *IEEE Computer 28*, 68–74.

Law, A. M. and W. D. Kelton (1991), *Simulation Modeling and Analysis*, Mc-Graw Hill.

Mah, B. A. (1997), "An empirical model of HTTP network traffic," In *Proceedings of the IEEE International Conference on Computer Communication (INFOCOM'97)*, IEEE Computer Society Press, Kobe, Japan.

Mosedale, D., W. Foss, and R. McCool (1997), "Lesson learned administering Netscape's Internet site," *IEEE Internet Computing 1*, 2, 28–35.

Pirolli, P., J. Pitkow, and R. Rao (1996), "Silk from a sow's ear: Extracting usable structures from the Web," In *Proceedings of 1996 International Conference on Human Factors in Computing Systems (CHI'96)*, ACM Press, Vancouver, Canada, pp. 118–125.

Pitkow, J. (1997), "In search of reliable usage data on the WWW," In *Proceedings of 6th International World Wide Web Conference (WWW6)*, Santa Clara, CA.

W3C (1995), "Common Logfile Format," http://www.w3.org/Daemon/User/Config/Logging.html.

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