Efficient State Estimators for Load Control Policies in Scalable Web Server Clusters*

V. Cardellini, M. Colajanni
University of Rome – Tor Vergata
Roma, I-00133
{cardellini, colajanni}@uniroma2.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 directing requests to the best server. One common approach is to use the Domain Name Server (DNS) as a centralized scheduler. However, address caching mechanisms and the non-uniformity of the load from different client domains complicate the load balancing issue and make existing scheduling algorithms for traditional distributed systems not applicable to Web server clusters. In this paper, we consider the theoretical DNS policies that require some system state information. We extend them to realistic situations where state information needs to be estimated with low computation and communication overhead. We show that, by incorporating these estimators into the DNS policies, load balancing improves substantially, even if the DNS control is limited to a small portion of client requests.

1. Introduction

With the rapid growth of WWW traffic, most popular Web sites need to scale up their server capacities. The most promising approach is to preserve a virtual single interface (URL) 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, it requires a mechanism for assigning requests to the Web server that can offer the best service [5, 9, 11].

The assignment decision can be taken at the *TCP-router* level or at *Domain Name Server* (DNS) level. A round-robin DNS policy is implemented by the NCSA server [11] and the SWEB server [1]. Other theoretical DNS scheduling policies are proposed in [5, 6]. Some TCP-router solutions are described in [4, 9, 10].

In this paper we will focus on Web server clusters based on DNS mechanisms. This architecture does not present risks of bottleneck, and can 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 and WWW address caching mechanisms that let the DNS control only a very small fraction of the user requests. Such a limited control is a big obstacle to load balancing among the Web servers. Real trace data indicate that even small caching periods such as five minutes would reduce the DNS control to few percent of all requests reaching the Web server cluster. The peculiarities of this scenario make scheduling algorithms for traditional distributed systems inappropriate to Web clusters and motivate the study of new DNS allocation schemes that require additional system state information to control the load of the Web servers. In this paper, we start with studying scheduling policies that can perform well under theoretical conditions that the DNS could have immediate access to any necessary state information, such as the load status of the servers and the exact percentage of requests coming from each domain. Then, we focus on the operational aspects of these policies in the light of their application to a DNS architecture working in a realistic environment. Main requirements of policies and estimators to be actually applicable in the WWW environment are low computational complexity, and compatibility with existing Web standards and protocols. 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.

The paper is organized as follows. Section 2 discusses the WWW components from the point of view of the Web server cluster. Section 3 reviews the most promising DNS scheduling policies focusing on the state information they require. Section 4 discusses whether and how this information can be obtained at the DNS. Section 5 presents the experimental results. Section 6 concludes the paper.

2. Scalable Web Cluster

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 provide the same documents, and a *Cluster Domain Name Server* (CDNS) that translates the URL-name into the IP-address of one of the servers in the cluster. The role of IP-address resolver allows the CDNS to distribute the requests based on some optimization criterion.

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 sub-network behind these local gateways as *domain*. Generally speaking, if one ranks the popularity of domains by the frequency of their accesses to a Web site, the distribution of the number of clients in each domain is a function with a short head 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 [2]. Here, in the simulation study we assume that the (2500) clients are partitioned among the (50) connected domains based on a Zipf's distribution, i.e. a distribution where the probability of selecting the i-th domain is proportional to $1/i^{(1-x)}$ (we consider $x = 0$).

As the focus is on Web cluster performance, we did not model the Internet traffic [8], but we consider major components, such as the name servers, that impact the performance of the cluster. Moreover, we consider all the details concerning a client *session*, i.e. the entire period of access to the Web site from a single user. 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; the *page 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. If the cache of the local name server has a valid mapping for this URL-name, the page request is sent directly to the Web server without passing through the CDNS. 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 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.

In the simulation, the number of page requests per session and the time between two page requests from the same client are assumed to be exponentially distributed with means 12 and 25 sec, respectively. 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*. The number of hits per page are obtained from a uniform distribution in the discrete interval (5—15). The hit service time and the inter-arrival time of hit requests to the server are assumed to be exponentially distributed with means 1/215 sec/hit and 0.25 sec, respectively. The cluster average utilization is always kept to 2/3 of the whole capacity in all experiments. This value is obtained as a ratio between *system load*, i.e. the total number of hits per second arriving to the Web cluster, and the *cluster capacity* which is the sum of the capacities of the single servers denoted in served hits per second. A thorough sensitivity analysis shows that main conclusions of the experiments are not affected by the choice of system parameters. On the other hand, the performance is very sensitive to connections from new domains and variations in the number of client accesses (see Section 5).

3. DNS Policies for Load Control

The CDNS has to address various issues that make it different from a normal scheduler and cause huge obstacles to the load balancing of the Web servers. We have already discussed about the non-uniform distribution of client requests among the domains and the control limited to a small fraction of the requests reaching the Web cluster. Other requirements that constrain the potential CDNS scheduling algorithm alternatives are the low computational complexity because scheduling decisions are required in real-time, and full compatibility with existing Web standards and protocols. In particular, all state information needed by a policy has to be accessible on the CDNS through some entities residing at the CDNS itself and/or at the Web servers. Any state information that needs some active cooperation from any other WWW components, such as browsers, name servers, clients, is not considered.

Scheduling policies such as round-robin and random do not require any state information. However, it has been shown in [5] that these algorithms perform very poorly under realistic scenarios when the clients are not uniformly distributed among the domains, and the name servers cache the URL-name to IP-address mapping for $TTL > 0$. 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 theoretical results described in [5, 6] which are of interest to 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 Web cluster make the 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. The key goal is to obtain an estimation of the so called *hidden load weight*, i.e. the average number of requests or hits that each domain sends to a Web server during a TTL interval after a new IP-address request has reached the CDNS.
- The CDNS has not to assign requests to already over-utilized servers. To this purpose, it is useful to combine the hidden load weight information with a simple mechanism that monitors the actual load of each server and checks whether it has exceeded a given *load threshold*. In this case, a critically loaded server sends an *alarm signal* to the CDNS that excludes it from any further assignment until its load falls below the threshold. We assume that all of following scheduling algorithms apply this feedback 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 promising algorithms with constant TTL [5] and one with adaptive TTL [6].

Two-tier Round-Robin (RR2). This algorithm is based on the consideration that the risk of overloading some of the servers is typically due to the requests coming from a few very popular domains. Therefore, RR2 uses the hidden load weight information to partition the domains connected to the Web cluster into two classes: *normal* and *hot* domains. In particular, RR2 sets a *class threshold* and evaluates the *relative hidden load weight*, which is with respect to the total number of requests in a TTL interval from all connected 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.

Dynamically Accumulated Load (DAL). This algorithm is a direct application of the hidden load weight information. 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 to predict how many requests will arrive to the chosen server due to this mapping. 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. The CDNS maintains an *assignment table* containing all 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.

Adaptive TTL (AdpTTL). This is a different class of algorithms that explore the TTL component that is chosen by the CDNS. The motivation for this approach comes from the observation that the number of page requests following an IP-address request increases with the TTL value. However, the naive solution of a simple reduction of the TTL value with the purpose of giving more control to the CDNS does not work. The basic idea of AdpTTL is to use any of the previous algorithms to select the Web server and then assign a different TTL to each IP-address request. The value can be tailored on the basis of the domain hidden load weight (and also of the server capacity if the cluster consists of heterogeneous Web servers [6]). The AdpTTL algorithm considered in this paper uses a RR2 policy to choose the IP-address of a Web server, and assigns as TTL a value inversely proportional to the hidden load weight of the domain from which the IP-address request has been issued.

The *constant TTL* algorithms use a TTL set to 240 seconds, while the *adaptive TTL* policy uses TTL=60 seconds for the hottest domain and higher values (reaching even 3000 seconds) for the other domains.

4. Heuristics for Estimating State Information

4.1. Information for hidden load weight estimation

All the proposed scheduling policies in Section 3 have to evaluate the hidden load weight of each connected domain. The most precise expression for this load is a function of the total number of hits issued to a server from a domain D during a TTL period that is,

$$W_{TTL}^{D} = \sum_{i=1}^{S^D} \sum_{j=1}^{R_i^D} h_{i,j}^{D} \qquad (1)$$

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 information coming from the clients to the CDNS is very limited because the source of an IP-address request is the only information that the CDNS gets without any overhead. Letting the CDNS collect the number of IP-address requests originated by each domain and, on this basis, estimate the hidden load weight does not require any additional communication, but it does not provide a good estimate. Examining Equation (1), it should be obvious that many of the parameters are not available on the CDNS. Various experiments confirm that, without additional state information to the CDNS, the cluster would not perform well due to the poor quality of the estimation of the hidden load weight.

A more 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 hidden load weight requires a periodic exchange of messages from the Web servers to the CDNS. The load information can be collected 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. All of them are based on the analysis of the *logfile* that each server maintains to keep trace of the client accesses. According to the Common Logfile Format [7], the information is reported for each hit. It 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.

The mean number of sessions (*WS.ses*) from each domain is a rough approximation of the hidden load weight per domain. This view of Equation (1) assumes that the 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 straightforward unless one uses a *cookie generation* mechanism or some heuristics that infers it through the site content topology or referred information [14]. For example, the algorithm proposed in [12] checks if a requested page can be directly reached from the already visited pages.

The mean number of page requests (*WS.req*) from each domain can be measured for each domain by excluding non-HTML requests from the counting. Implicitly, this approximation of the hidden load weight assumes that the average number of hits per request is similar for all domains.

As shown in Equation (1), estimating the workload as a function of the number of hits (*WS.hit*) from each domain is the most accurate way to obtain the hidden load weight. However, even this measure of the load has some potential weakness. 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 [13].

4.2. Heuristic analysis

We focus on heuristics that can be useful 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* is superior to other approaches when the system is subject to non-stationary load behavior, its computational complexity and the very large sample sizes 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*. 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 hidden load weight for a domain D 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. The observed requests can be sessions, pages or hits.

5. Performance Results

The main goal of this study is to investigate the impact of the CDNS scheduling algorithms on avoiding that some server becomes overloaded. 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. If their 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 problem at t_2.

Specifically, the major performance criteria 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.

We compare the performance of the scheduling policies combined with some heuristics for state estimate against the theoretical versions of the same algorithms under a *static* and *dynamic* scenario for different load conditions. (A *theoretical algorithm* knows the load status of the servers and the exact hidden load weight of each domain).

A *static scenario* is an environment in which the requests arrive from a given set of domains each with a given request rate during the entire period of the simulation study.

We first verified how well the heuristics in Section 4 estimate the *relative hidden load weights* in a static environment. The experiments (not shown due to space limits) revealed that the estimation accuracy is insensitive to the value of the sampling period when this is larger than 200 seconds [3]. Another interesting result is that, at least for the estimation of the relative hidden load weight, there is not much difference in using a simple mean, a weighted mean on two intervals, and a weighted mean on four intervals [3]. The DNS information alone does not estimate well the relative hidden load weight per each domain. On the other hand, any of the *WS.ses*, *WS.req* and *WS.hit* approaches can provide an accurate estimation for any long-tail distribution of the clients among the domains. The good estimate of the hidden load weight leads the performance of the three policies with a constant TTL (that is, RR2, DAL, and MRL) to be quite close to that obtained by the theoretical algorithm. However, it is the AdpTTL policy with adaptive TTL that achieves the best performance. A CDNS using this scheduling algorithm provides, with probability close to 1, a utilization lower than 0.9 for all servers. On the other hand, conventional policy (such as RR) which does not consider any state information performs very poorly.

A *dynamic scenario* increases the difficulties of the heuristics to obtain accurate estimates of the hidden load weight. 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. In other words, we assume a modification of the hidden load weight behind 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 reach suddenly the top 10 ranks.

Figures 1–4 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. We use the *simple mean* on one sampling period of 240 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 round-robin results.

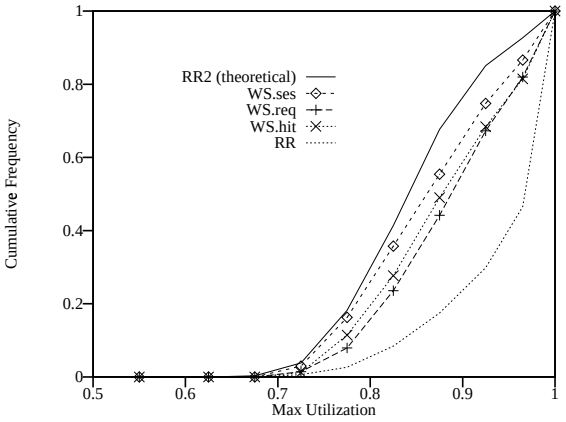

Figure 1. Performance of RR2 policy.

Comparing the performance under the different heuristic estimation algorithms, we see that the *WS.hit* estimate provides the best performance under MRL and AdpTTL policies (Figure 3 and 4, respectively). For the RR2 policy, the *WS.ses* estimate gives the best results (Figure 1), while for the DAL policy there is no appreciable difference among the three load estimates (Figure 2). We can summarize that MRL is the policy that has the closest results to its theoretical version. However, AdpTTL is still the policy that

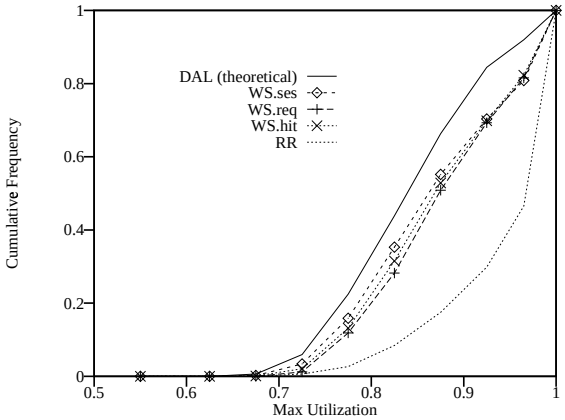

Figure 2. Performance of DAL policy.

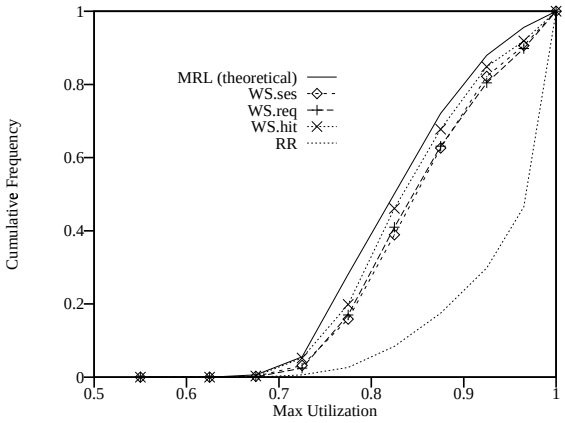

Figure 3. Performance of MRL policy.

performs the best even if its results are not as close to the theoretical version of this algorithm.

In the last set of experiments we evaluate the sensitivity of the scheduling policies and related heuristic on state estimate to the *inter-rerank time* under different sampling periods. The load information used for Figure 5–7 is based on *WS.hit*. For this set of results, a different performance metric is used where we show the 96 percentile of the cluster maximum utilization. In other words, the probability that no server of the Web cluster is overloaded (or exceeding 96% utilization) is considered as the metric. 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).

Figure 5 and 6 show that all scheduling policies with a constant TTL are quite robust (i.e. the probabilities of overloading are never above 0.8) and in most cases 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). On the other hand, the AdpTTL

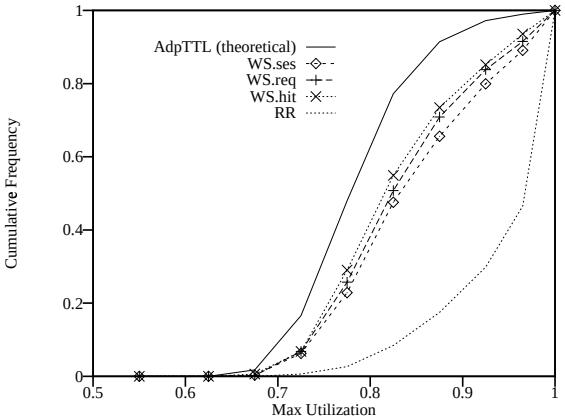

Figure 4. Performance of AdpTTL policy.

is much more sensitive to the variability of the scenario on inter-rerank time. The performance improves as the inter-rerank time increases. We can conclude that although AdpTTL policy usually offers the best performance, it requires a more precise estimation of the hidden load weight.

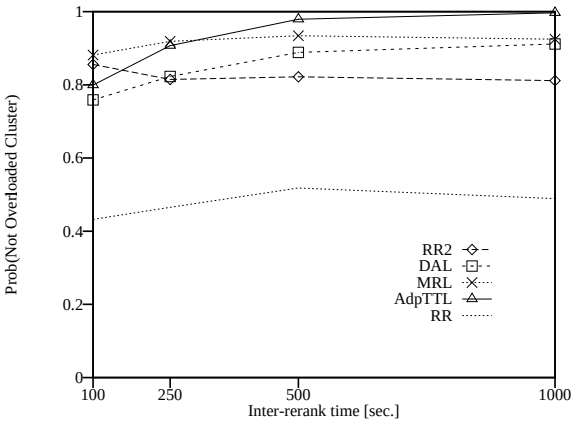

Figure 5. Sensitivity to the domain inter-rerank time (simple mean on one sampling period of 240 sec).

These results motivate the search for a more accurate heuristics that can work well even if the inter-rerank is as low as 100 seconds. To the purpose of fairly comparing the *simple mean* to the *weighted mean* approach, we consider in Figure 7 a sampling period on 480 seconds with a decay distribution applied to four sub-intervals of 120 seconds. A comparison between this figure and Figure 6 shows that applying a weight on the collected state data tends to give more stable result when the sampling period is relatively large (480 seconds or more) and it is especially beneficial to the AdpTTL policy.

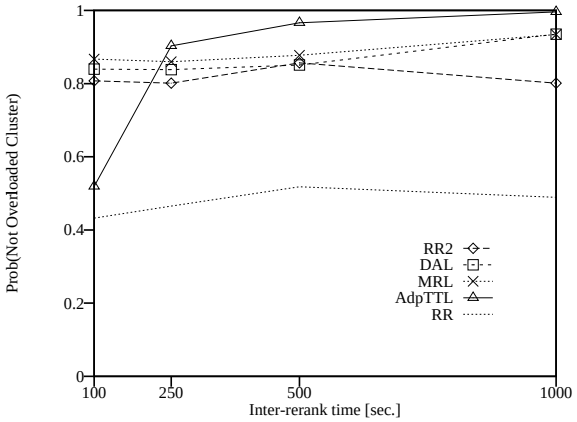

Figure 6. Sensitivity to the domain inter-rerank time (simple mean on one sampling period of 480 sec).

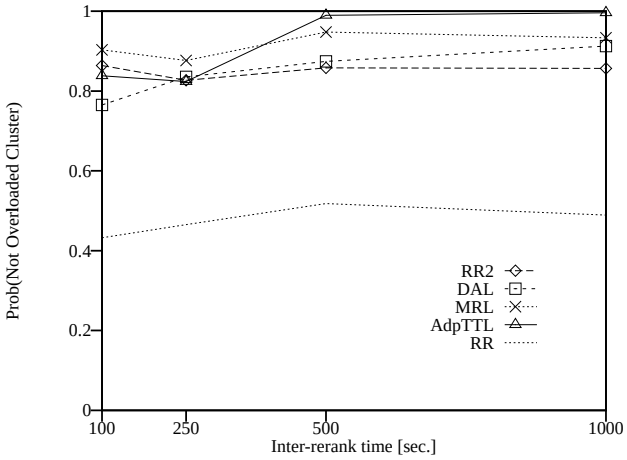

Figure 7. Sensitivity to the inter-rerank time of the domains (decay distribution on four sampling periods of 120 sec. each).

6. Conclusions

Replication of information among multiple Web servers is necessary to support high request rates to popular Web sites. In this paper we have studied a Web cluster architecture in which the CDNS also plays the role of a centralized scheduler. We move from the consideration that a CDNS scheduler needs some state information to take decisions that avoid overloading the Web servers. In particular, we examine the policies that perform well under theoretical conditions where all state information is readily available and extend them to realistic situations where 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, in particular the hidden load weight for each domain. 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. In particular, the adaptive TTL algorithm gives the best results even if it is the most sensitive to the quality of the heuristics.

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. Arlitt, C.L. Williamson, "Web server workload characterization: The search for invariants", *IEEE/ACM Trans. on Networking*, vol. 5, no. 5, pp. 631–645, Oct. 1997.

[3] V. Cardellini, M. Colajanni, P.S. Yu, "Efficient state estimators for load control in scalable Web server clusters", *IBM Res. Rep. RC 21085*, Yorktown Heights, NY, Jan. 1998.

[4] Cisco's LocalDirector http://www.cisco.com/warp/public/751/lodir/index.shtml

[5] M. Colajanni, P.S. Yu, D.M. Dias, "Analysis of task assignment policies in scalable distributed Web-server system", *IEEE Trans. on Par. and Distr. Systems*, vol. 9, no. 6, June 1998.

[6] M. Colajanni, P.S. Yu, V. Cardellini, "Dynamic load balancing in geographically distributed heterogeneous Web-servers", *Proc. of IEEE 18th Int. Conf. on Distributed Computing Systems*, Amsterdam, May 1998.

[7] Common Logfile Format http://www.w3.org/ Daemon/User/Config/Logging.html

[8] 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.

[9] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, "A scalable and highly available Web server", *Proc. of 41st IEEE Computer Society Int. Conf.*, pp. 85–92, Feb. 1996.

[10] G.D.H. Hunt, G.S. Goldszmidt, R.P. King, R. Mukherjee, “Network Dispatcher: A connection router for scalable Internet services”, *Proc. of 7th Int. World Wide Web Conf.*, Brisbane, Australia, Apr. 1998.

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

[12] P. Pirolli, J. Pitkow, R. Rao, “Silk from a sow’s ear: Extracting usable structures from the Web”, *Proc. of Int. Conf. on Human Factors in Comp. Systems*, Vancouver, Apr. 1996.

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

[14] K-L. Wu, P.S. Yu, A. Ballman, “Speed Tracer: A Web usage mining and analysis tool”, *IBM Systems Journal*, vol. 37, no. 1, pp. 89-105, 1998.