Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers*

Michele Colajanni†
Dip. di Informatica, Sistemi e Produzione
Università di Roma – Tor Vergata
Roma, Italy 00133
colajanni@uniroma2.it

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

Valeria Cardellini
Dip. di Informatica, Sistemi e Produzione
Università di Roma – Tor Vergata
Roma, Italy 00133

Abstract

With ever increasing Web traffic, a distributed multi-server Web site can provide scalability and flexibility to cope with growing client demands. Load balancing algorithms to spread the requests across multiple Web servers are crucial to achieve the scalability. Various domain name server (DNS) based schedulers have been proposed in the literature, mainly for multiple homogeneous servers. The presence of heterogeneous Web servers not only increases the complexity of the DNS scheduling problem, but also makes previously proposed algorithms for homogeneous distributed systems not directly applicable. This leads us to propose new policies, called adaptive TTL *algorithms, that take into account of both the uneven distribution of client request rates and heterogeneity of Web servers to adaptively set the time-to-live (TTL) value for each address mapping request. Extensive simulation results show that these strategies are robust and effective in balancing load among geographically distributed heterogeneous Web servers.*

†Supported by the Italian Ministry of University and Scientific Research in the framework of the project *High Performance Systems for Distributed Applications.*

1 Introduction

With the explosive growth of the World-Wide Web (WWW), the traffic on the popular Web sites is increasing rapidly. The Web servers often need to be based on a distributed or parallel architecture while preserving a virtual single interface. Some load balancing mechanisms are required to spread the request load among the multiple server nodes. The effectiveness of the load balancing mechanism is critical in determining the scalability of a Web site. In the WWW environment, each client request includes a Uniform Resource Locator (URL) name to identify the requested resource. The mapping of a URL name into the IP address of a Web server can be done through a separate *address mapping* request to the Domain Name Server (DNS). This makes DNS an ideal place to explore load balancing mechanism via address mapping to return the best candidate server location for retrieving a document [6, 4]. Examples of multiple Web servers using DNS routing are the NCSA HTTP-server [6] and the SWEB server [1].

To reduce the address request traffic, caching of the address mapping is typically done at Name Servers (NS) and also at the clients. A *time-to-live* (TTL) is decided by the DNS for each mapping such that address requests made within the specified TTL are resolved by the NS. However, this address caching mechanism only allows the DNS to directly route a very few percentage of the requests to the Web servers. This makes the DNS an atypical centralized scheduler, and opens entirely new problems because a centralized scheduler of traditional parallel/distributed sys-

tems has generally full control on the job requests.

In [4], the load balancing issue is examined for a distributed Web site consisting of multiple homogeneous servers. Here we examine dynamic load balancing among heterogeneous Web servers. The heterogeneity assumption, which is much more likely in a geographically distributed environment, introduces an additional degree of complexity to the request scheduling problem in distributed Web sites and strongly affects the applicability of the previously proposed load balancing algorithms for homogeneous systems. Other static and dynamic scheduling policies for heterogeneous parallel systems which are proposed in [7] cannot be used because of the peculiarities of the DNS scheduler. In this paper, a new class of algorithms is devised and evaluated to provide better load balancing among distributed heterogeneous Web servers. Our performance results show that the approach of adaptively setting the TTL value based on the client request rate and/or the server capacity can dramatically reduce system imbalance even in a highly heterogeneous Web site. The *adaptive TTL* schemes show high robustness on performance even in the presence of non-cooperative NS in setting the TTL value and inaccurate estimate of system state information which any DNS scheduler may inevitably encounter.

The paper is organized as follows. Section 2 presents a general model for heterogeneous distributed Web sites and focuses on the issues that this system introduces on the DNS scheduling problem. Section 3 presents the new class of *adaptive TTL* algorithms in deterministic and probabilistic form. Section 4 describes the model parameters and the performance metrics that are of interest for this paper. Section 5 presents the simulation results for a wide set of system configurations and scenarios.

2 System Model and DNS Scheduling Problem

We assume the model of a Web site consisting of N servers and a primary DNS that can translate a URL name into the IP-address of one of the Web servers. The set of $\{S_1, \ldots, S_N\}$ heterogeneous servers is numbered in decreasing processing capacity. Each server may consist of single or multiple processors, have a different architecture and disk speed. However, we only address heterogeneity through the notion that each server may have a different capacity to satisfy client requests. In particular, each server S_i is characterized by an *absolute capacity* C_i that is expressed as the number of requests per second it can handle, and a *relative capacity* $\xi_i = C_i/C_1$ that is, the ratio between its capacity and the capacity of the most powerful server among them. The requests come from client *domains* that have a (set of) local name server(s) and are connected to the network through local gateways such as firewalls or SOCKS servers.

Web sites that use a DNS scheduler for load balancing generally take the round-robin (RR) algorithm to map different client requests to the servers [6]. However, as a result of address caching, a large number of the subsequent client requests from a particular domain are mapped to the same server during the TTL period. This can lead to load imbalance among the servers, as quantified in [5]. In [4], it was found that RR works well only under the unrealistic hypothesis that all domains have the same client request rate. Conversely, if one ranks the popularity of domains by the frequency of their accesses to the Web site, the client request rate has a long-tail distribution. An analysis on academic and commercial Web sites shows that in average 75% of the client requests come from only 10% of the domains [2]. For this reason, we assume that clients are partitioned among the domains based on a Zipf's distribution that is, a distribution where the probability of selecting the i-th domain is proportional to $1/i^{(1-x)}$ [8].

One important consideration in dealing with the problem of nonuniform distribution of requests and limited control of the DNS is the kind of information that is needed by the scheduler. For homogeneous servers in [4], it was found that the following actions improved performance: estimating the average request rate of each connected domain; identifying the source domain of the client requests in making scheduling decision; identifying critically loaded servers and temporarily excluding them from scheduling consideration. In this paper, we explore strategies that exploit these kinds of information.

The first goal is to estimate the average number of client requests from each domain during a TTL interval. This value, referred to as the *hidden load weight* since it is invisible to the DNS, denotes the average number of requests issued from each domain to a server after an address mapping decided by the DNS. Various scheduling algorithms that take into account of hidden load weight and source domain address of requests have been proposed and evaluated in [4] for a homogeneous distributed server environment, e.g. *minimum dynamically accumulated load* (DAL), *minimum residual load* (MRL), *two-tier Round-Robin* (RR2). In this paper, we focus on applying the adaptive TTL concept to the *two-tier Round-Robin* (RR2) policy that generally gives satisfactory result in homogeneous systems, and is most adaptable to hetero-

geneous distributed systems. The RR2 is based on the following simple concept. Since the clients are unevenly distributed among the domains, the hidden load weight of each domain is typically very different. Therefore, this policy assumes a partition of the domains into two classes: *normal* domains, and *hot* domains. A *class threshold* χ is introduced such that each domain with a relative hidden load weight greater than χ is included in the hot class. For each address request, RR2 first determines the class of the source domain and then selects the next server in round-robin order following the server scheduled for the last request in that class. The objective is to reduce the probability that requests from the hot domains are assigned too frequently to the same server.

Even in a homogeneous server environment, it is hard for the DNS to estimate the relative load condition at each of the Web servers based on its previous scheduling history. The hidden load weight is only an estimate of the potential impact of each scheduling decision, while the number of requests implied by the hidden load spread over a period of time. Therefore, to be sure that the server chosen by the DNS is not too busy, it is useful to use a simple asynchronous feedback mechanism that monitors the actual load condition on the servers. Each server periodically calculates its utilization and checks whether it has exceeded a given *alarm threshold* ϑ. When this occurs, the server sends an alarm signal to the DNS, while a normal signal is sent when its utilization level returns below the threshold. We assume that all of scheduling algorithms to be discussed next use this type of asynchronous feedback mechanism and consider a server as a candidate for receiving requests only if that server has not declared critically loaded.

3 DNS Scheduling Algorithms

Straightforward modification of the policies for a homogeneous multi-server Web site [4] is inadequate to handle both system heterogeneity and nonuniform client distribution as will be in part shown later in Figure 3. Various experiments (not reported due to space limits) showed that other proposed policies are not adaptable to a heterogeneous environment and hence not pursued further in this study. These results motivated the search for an alternative approach. Looking at DNS activities, we observe that when an address request reaches the DNS, the scheduler returns not only the IP address of the chosen Web server but also the period (TTL interval) during which this address mapping is valid. Hence, the proposed class of scheduling algorithms explores the idea of adaptively adjusting the TTL value to facilitate load balancing. The goal is not to simply reduce the TTL value in order to give more control to the DNS. Using a TTL value close to 0 would give full control to the DNS. However, very small TTL values are typically ignored at NS to avoid overloading the network with name resolution traffic. Another important reason is that a naive reduction of TTL value does not reduce the load skew due to unevenly distributed client request rates.

The basic idea here is to assign to each address request a different TTL value by taking into account not only the data request rate of the source domain originating the request, but also the capacity of the server chosen by the DNS. By properly selecting the TTL value for each address request, we can reduce the load skews that are the main cause of overloading, especially in a heterogeneous system. The objective of the proposed approach is to even out the impact of the subsequent requests during the TTL interval on each server. More specifically, we want the subsequent requests from each domain to consume similar amount of server utilization or percentage of server capacity. This can address both server heterogeneity and nonuniform client rates. The new class of scheduling disciplines that use this approach is called *adaptive TTL*. We combine the adaptive TTL concept with the basic RR algorithm and its RR2 variant. Furthermore, we consider the *deterministic* and *probabilistic* versions of these algorithms for the selection of the Web server.

3.1 Probabilistic Algorithms

RR and RR2 policies can be straightforwardly extended to a heterogeneous Web site through the addition of some *probabilistic* routing features. The basic idea is to make the round-robin type assignment probabilistic based upon the server capacity. For this purpose, we generate a random number γ ($0 \leq \gamma \leq 1$) and, under the assumption that S_{i-1} was the last chosen server, we assign the new requests to S_i only if $\gamma \leq \xi_i$. Otherwise, we skip the server S_i and consider S_{i+1} repeating the same process. This straightforward modification allows RR and RR2 to schedule the requests taking into account of the various server capacities. The probabilistic versions of these algorithms are denoted by PRR and PRR2, respectively.

Once the server has been selected, the TTL value is assigned based on the domain of the request. This method assumes that the hidden load weight of each domain can be dynamically estimated. This can be done by having the servers keep track of the number of incoming requests from each domain and the DNS periodically collect the information and calculate the client request rate from each domain. In its most generic form, we denote by TTL/i the policy

that partitions the domains into i classes based on the hidden load weight and assigns a different TTL value to address requests arriving from a different domain class. TTL/i is a meta-algorithm that includes various strategies. For $i = 1$, we obtain a degenerate policy (TTL/1) that uses the same TTL for any domain, hence not an adaptive TTL algorithm; for $i = 2$, we have the policy (TTL/2) that partitions the source domains into *normal* and *hot* domains, and chooses a high TTL for requests coming from normal domains, and a low TTL for requests coming from hot domains. Analogously, for $i = 3$, we have a strategy that uses a three-tier partition of the domains, and so on, until $i = K$ that denotes the algorithm (TTL/K) that uses a different TTL for each domain. For TTL/K policies, let TTL_j denote the TTL value chosen for the requests from the j-th domain,

$$TTL_j = \frac{\lambda_{max}\overline{TTL}}{\lambda_j}$$

where $\overline{TTL}$ is the minimum TTL value used by the DNS, λ_j and λ_{max} are the relative hidden load weights of the j-th domain and the most popular domain, respectively.

3.2 Deterministic Algorithms

The server selection is done through the traditional RR or RR2 policy. Unlike probabilistic disciplines, deterministic algorithms take into account of both nonuniform request rates and server heterogeneity on setting the TTL. The approach is similar to that described for the probabilistic disciplines. However, the TTL value is now chosen by considering the server capacity as well. For the generic TTL/S_i policy, we partition the domains into i classes, and estimate the average hidden load weight for each class. The TTL for each class and server is set inversely proportional to the class weight while proportional to the server capacity. The deterministic TTL/S_1 algorithm is a degenerate case that considers server heterogeneity only and ignores the domain of client requests. The TTL/S_2 policy uses two TTL values for each server depending on the source domain of the requests, i.e. normal or hot domain. The TTL/S_K algorithm selects a TTL value for each server and domain combination. Specifically, let TTL_{ij} be the TTL chosen for the requests from the j-th domain to the i-th server,

$$TTL_{ij} = \frac{\lambda_{max}\overline{TTL}}{\lambda_j}\xi_i\eta$$

where $\eta = C_1/C_N$ is the *processor power ratio* that measures the degree of heterogeneity of the distributed Web architecture [7].

Similarly to probabilistic disciplines, we denote by DRR and DRR2 the traditional RR and RR2 policies when applied in combination with adaptive TTL values. Note that DRR-TTL/S_1 (respectively, DRR2-TTL/S_1) uses a different TTL value for each server capacity, whereas PRR-TTL/1 (respectively, PRR2-TTL/1) uses a single constant TTL.

4 Performance Analysis

4.1 System parameters

The focus of our study on the Web site throughput allows us to avoid the details of the network architecture and the Internet traffic. On the other hand, we model various details of a client session characterized by one address resolution, and several data retrievals. The client obtains through the DNS scheduler or some (client's or NS's) cache a mapping to one of the servers. Since the Web site consists of servers with identical information, the requests can be assigned to any of the servers. Once the client has been connected to the server, the client is modeled to submit multiple page requests. Each of them actually consists of a burst of requests (*hits*) representing the HTML page and the objects contained in it. The number of hits per page are obtained from a uniform distribution in the discrete interval (5—15). The number of page requests per session and the time between two page requests from the same client (*mean think time*) are assumed to be exponentially distributed [2].

The parameters to be considered in our simulations fall under five categories. They are presented in Table 1 with their default values shown between parentheses. We assume that clients are partitioned among the K domains on a pure Zipf's distribution basis [8]. We use the maximum difference between the relative server capacities to denote the four levels of server heterogeneity considered in the study. Assuming $N = 7$, the relative server capacities $\{\xi_i's\}$ are given in Table 2 for the four cases. To allow a fair comparison among the performance of these systems, we keep constant the total system capacity (1500 hits per second) and the average system utilization (2/3 of the total capacity).

The parameters for the scheduling algorithms are chosen as follows. Each server calculates its utilization every 8 seconds, and sends an alarm signal to the DNS if this value has exceeded the alarm threshold ϑ. Moreover, the *constant TTL* and *probabilistic TTL/1* algorithms use a single TTL value of 240 seconds. Since an arbitrary choice of TTL would lead to unfair performance comparisons, for each adaptive TTL policy we have chosen the TTL values in such a way that their average address request rates remain the same.

The simulators were implemented using the CSIM package. Each simulation run is made up of five hours

Category	*Parameter*	*Setting (default)*
Domain	Connected	K=10–100 (20)
	Clients per domain	pure Zipf's
Client	Total number	1500
	Mean think time	10–30 sec (15)
Request	Requests per session	20 pages
	Hits per request	(5–15)
Web site	Servers	N=5–7 (7)
	Total capacity	1500 hits/sec
	Heterogeneity	0–65%
	Average utilization	0.6667
Algorithm	Utilization interval	8 sec
	Alarm threshold	$\vartheta = 0.75$
	Class threshold	$\chi = 1/K$
	Constant TTL	240 sec

Table 1: Parameters of the system model

Heterogeneity Level	*Relative Server Capacities (ξ_i)*
20%	$\{1, 1, 1, 0.8, 0.8, 0.8, 0.8\}$
35%	$\{1, 1, 0.8, 0.8, 0.65, 0.65, 0.65\}$
50%	$\{1, 1, 0.8, 0.8, 0.5, 0.5, 0.5\}$
65%	$\{1, 1, 0.8, 0.8, 0.35, 0.35, 0.35\}$

Table 2: Parameters of the heterogeneity levels

of the Web site activities. Confidence intervals were estimated, and the 95% confidence interval was observed to be within 4% of the mean.

4.2 Performance metrics

The main goal of this study is to investigate the impact of the DNS scheduling algorithms on avoiding that any server becomes overloaded. For this reason, we do not adopt traditional metrics such as the standard deviation of server utilizations. Here, the performance of the various policies is evaluated through the maximum server utilization observed during the simulation run. The main performance metric reported is the cumulative frequency of the maximum utilization among the servers (*Max Utilization*), i.e. we present for each level of utilization the probability (or fraction of time) that all server utilizations stay within that level. This metric provides an indication on the relative frequency of overloading. For example, if the probability of all servers less than 80% utilized is 0.75, it implies that the probability of at least one server exceeding 80% utilized is 0.25.

We report a different metric when we evaluate the performance of the algorithms as a function of system parameters, such as server heterogeneity and number of domains. In these graphs, we show the 98 percentile of the maximum server utilization that is, the $Prob(MaxUtilization < 0.98)$.

5 Performance Results

5.1 Adaptive TTL schemes

The first set of simulation results evaluate how system heterogeneity affects the performance of the algorithms that use *adaptive TTL* policies. Figure 1 compares various deterministic algorithms for a low level of system heterogeneity (20%). In this figure, we also report the *ideal* case that is, the PRR policy applied under uniform distribution of client requests. The conventional RR policy used in [1, 6] represents the lower-bound case. The y-axis is the cumulative probability of the maximum server utilization reported on the x-axis. The higher the probability the less likely that some of the servers will be overloaded, hence better load sharing is achieved. For example, under the DRR2-TTL/S_K policy, the probability of having maximum utilizations less than 0.9 is 0.94. For RR, this probability is only around 0.1. That is to say for RR, 90% of the time at least one of the servers is above 90% utilization, while under DRR2-TTL/S_K, this reduces to only 6% of the time. However, taking only into account of server heterogeneity in setting the TTL value, as done by deterministic TTL/S_1 schemes, does not improve performance much with respect to RR. The probability of having maximum utilization less than 0.9 is still less than 0.35. Conversely, all schemes that address both client skew and server heterogeneity perform significantly better than TTL/S_1 policies. In particular, the strategies that use a different TTL for each server and domain combination, namely DRR-TTL/S_K and DRR2-TTL/S_K, have results close to the envelope curve of the ideal case. RR and RR2-based strategies have similar performance, even if the latter are always better than the former policies. (Note that the difference is more significant for homogeneous servers with constant TTL as reported in [4].)

Analogous results are achieved by the probabilistic schemes that combine *adaptive TTL* to handle nonuniform client distribution and probabilistic routing to address system heterogeneity. Figure 2 shows the cumulative probability of their maximum server utilization for a heterogeneity level of 35%. The relative order among the strategies remains the same as in Figure 1. Generally speaking, the RR2-based policies are slightly better than the RR-based strategies. Moreover, even the PRR-TTL/2 strategy performs consistently better than the PRR-TTL/1 algorithm that simply generalizes the conventional RR scheme to heterogeneous servers. This demonstrates that probabilistic routing alone cannot handle the nonuniform client distribution.

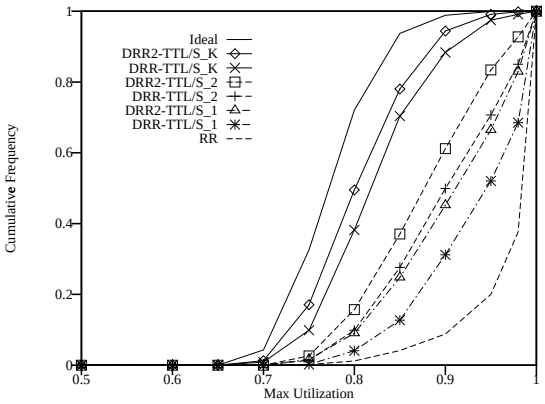

Figure 1: *Deterministic* algorithms (Het. 20%)

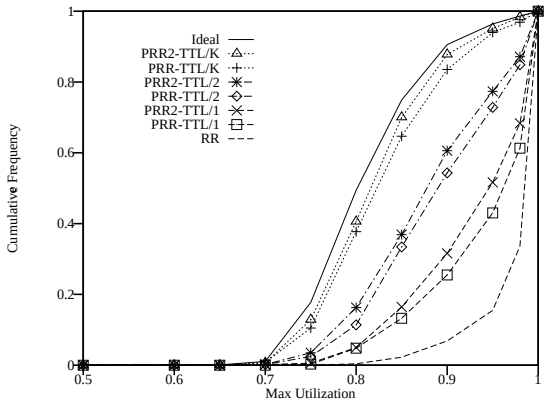

Figure 2: *Probabilistic* algorithms (Het. 35%)

Figure 3 analyzes the sensitivity of the deterministic and probabilistic RR2-based policies to the system heterogeneity that varies from 20% to 65%. Moreover, to demonstrate that other schemes proposed for homogeneous Web servers [4] are not applicable to a heterogeneous system, this figure shows also the performance of the DAL policy (in a version that takes into account the different capacity of the servers). Now, the y-axis is the probability that the maximum server utilization is less than 0.98, while the x-axis denotes the level of system heterogeneity. This figure shows that adaptive TTL algorithms are relatively stable, i.e. their performance does not vary much until the system heterogeneity exceeds 50%. After this level, especially the TTL/2 and TTL/S_2 schemes do not perform very well. Moreover, TTL/K and TTL/S_K are very effective in almost always avoiding overloading any of the servers as the probability shown is so close to 1. While achieving better performance, these algorithms also display better stability compared to other adaptive schemes. Hereafter, we do not consider the disciplines with single TTL for each server (that is, probabilistic TTL/1 and deterministic TTL/S_1 strategies), because of their instability and poor performance.

The shown results and other experiments not reported here due to space limitations lead to the following conclusions.

- Varying TTL as a function of server capacity and domain request rate substantially improves the performance of the DNS scheduling algorithms. *Adaptive TTL* schemes achieve much better load balancing than any *constant TTL* strategy. Especially TTL/K and TTL/S_K algorithms perform fine even when the system is highly heterogeneous and client requests are unevenly distributed among domains. This result is somewhat surprising, as the DNS scheduler has direct control over a very limited fraction of requests (the percentage is often below 4%).

- Deterministic policies work typically better than probabilistic schemes. However the difference is not large and tend to diminish when the system heterogeneity increases.

- Maintaining two-tier scheduling for differentiating requests from popular and normal domains still yields positive effect on the performance. Indeed, RR2-based strategies always perform better than their RR-based counterpart.

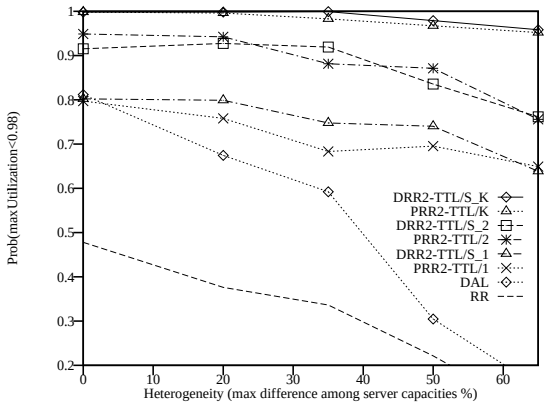

Figure 3: Sensitivity to system heterogeneity

5.2 Robustness of adaptive TTL schemes

We now examine the robustness of the adaptive TTL schemes. Two specific aspects are considered. One is on the impact of NS not following the recommended TTL value. The other is on how sensitive the performance is to the accuracy of the estimated

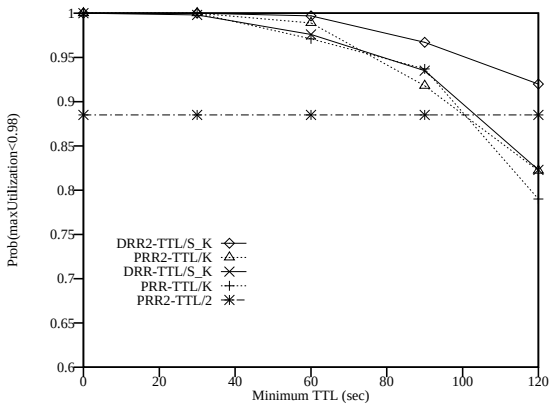

Figure 4: Sensitivity to minimum TTL (Het. 20%)

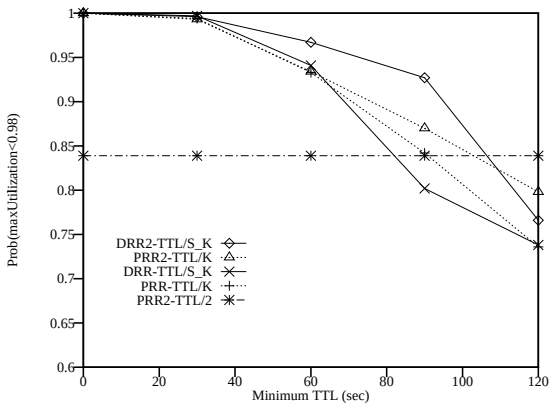

Figure 5: Sensitivity to minimum TTL (Het. 50%)

hidden load weight. The latter is less of an issue if the load from each domain remains relatively stable or changes slowly. However, in a more dynamic environment where client request rates from the domains may change constantly, it can be difficult to obtain an accurate estimate.

Each NS caches the name-to-address mapping for the TTL period or for a default value if the decided TTL is considered too small. Since there does not exist a common TTL lower bound which is accepted by all NSs, in our study we consider the worst case scenarios, where all NSs become non-cooperative if the proposed TTL is lower than a given minimum threshold, and perform sensitivity analysis against this threshold.

Figure 4 shows the sensitivity to the minimum TTL threshold of the four TTL/K and TTL/S_K variants and PRR2-TTL/2 algorithm for a system heterogeneity level of 20%. DRR2-TTL/S_K always performs the best. Furthermore, PRR2-TTL/2 is very much insensitive to the minimum accepted TTL. The advantage from DRR2-TTL/S_K reduces as the minimum TTL value increases because this scheme may sometimes need to select a low TTL value when a client request coming from a hot domain is assigned to a server with limited capacity. Conversely, the PRR2-TTL/2 strategy is almost not affected by the problem of non-cooperative NS because it uses a rough partitioning (i.e. two classes) of the domains and is able to always assign TTL higher than 180 seconds in all experiments.

Figure 5 considers the same set of policies in Figure 4 under a system heterogeneity level of 50%. Now, DRR2-TTL/S_K still provides the best performance, if the minimum TTL threshold is less than 100 seconds. Otherwise, PRR2-TTL/2 becomes the algorithm that performs the best, while the PRR2-TTL/K strategy also performs better than DRR2-TTL/S_K for rather high TTL threshold values.

Next we consider the issue of the accuracy of the parameter estimation on domain loads. (Further work on efficient parameter estimation can be found in [3].) We examine how the maximum error in estimating the domain load may affect the performance. Figures 6 and 7 compare some representative adaptive TTL schemes as a function of the estimation error, when the system heterogeneity is 20% and 50%, respectively. In the experiment, we introduce a perturbation to the request rate of each domain, while the DNS estimates of the hidden load weight remain the same as before. For the case of a ψ% error, the request rate of the busiest domain is increased by ψ% and the request rates of the other domains are proportionally decreased to maintain the same total request rate. This effectively increases the skew of the client rate distribution, hence represents a worst case.

When the estimation error or load perturbation increases, the system performance decreases in all eight algorithms. However, all the TTL/K and TTL/S_K schemes clustered on the top of the figure show much less sensitivity than the TTL/2 and TTL/S_2 schemes on the bottom. In particular, when the server heterogeneity is high ($\geq$ 50%) and the error is large ($\geq$ 30%), the performance of TTL/2 and TTL/S_2 strategies can degrade substantially. This is in contrast to the TTL/K and TTL/S_K schemes which are only slightly affected by the error in estimating the domain load. When the system heterogeneity is less than 50%, performance of these schemes degrades at most a few percentage point, compared to the case with no estimation error. This shows the high robustness of the TTL/K and TTL/S_K algorithms.

In summary, we found that when there is full con-

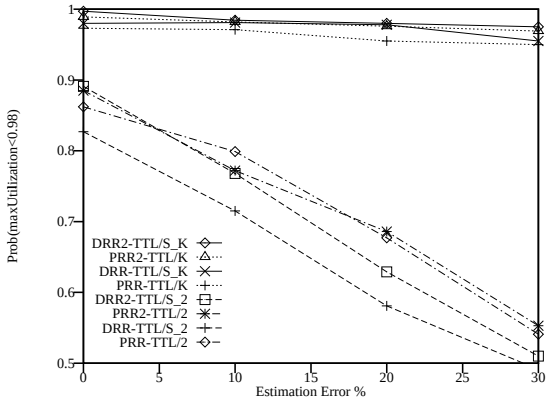

Figure 6: Sensitivity to error in estimating the domain *hidden load weight* (Het. 20%)

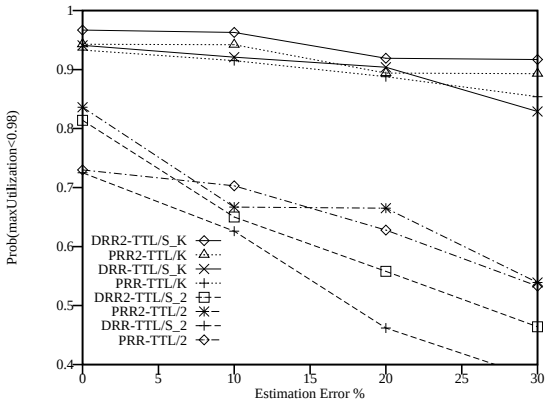

Figure 7: Sensitivity to error in estimating the domain *hidden load weight* (Het. 50%)

trol on the choice of the TTL values, DRR2-TTL/S_K is the strategy of choice. Conversely, when there are many non-cooperative name servers imposing their own minimum TTL thresholds, DRR2-TTL/S_K still provides the best performance, if the system heterogeneity is low (less than 30%) and the accepted minimum TTL values are moderate (below 120 seconds). Otherwise, if both the TTL threshold and system heterogeneity are high, a TTL/2 scheme typically performs better. When the accepted TTL threshold is high (more than 120 seconds) PRR2-TTL/K can provide better results than DRR2-TTL/S_K under high system heterogeneity.

Finally, the adaptive schemes that assign a different TTL value to each connected domain perform well even in the case when the domain load cannot be estimated accurately or the client request rates are highly variable. This is especially the case for DRR2-TTL/S_K and PRR2-TTL/K.

6 Conclusions

Although geographically distributed multi-server Web sites have the potential to greatly improve throughput performance, their success critically depends on load sharing algorithms to support scalability. Many scheduling algorithms for conventional parallel and distributed systems have previously been proposed and analyzed. However, none of them can be directly applied for dynamic load scheduling in distributed Web sites. The main problems are that the DNS scheduler can only explicitly route a small fraction of the requests, and these requests are unevenly distributed among the domains. The problems are further complicated when we consider a distributed Web site consisting of heterogeneous servers with different capacities.

We found that straightforward modification of known scheduling strategies or those developed for the homogeneous systems [4] gave unsatisfactory results. In this paper, the new class of *adaptive TTL* strategies is proposed. These schemes assign a different TTL value to each address request by taking into account the capacity of the selected server and/or the request rate of the source domain of the request. *Adaptive TTL* strategies show low computational complexity and high robustness, and do not require many system state information. Such characteristics make this class of policies an ideal candidate for the Internet environment, which is subject to heterogeneous servers with intrinsic high load skews and dynamic variations.

References

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

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

[3] V. Cardellini, M. Colajanni, P.S. Yu, "Efficient state estimator for load control in scalable Web server clusters", IBM Research Report RC 21085, Yorktown Heights, NY, 1998.

[4] M. Colajanni, P.S. Yu, D.M. Dias, "Scheduling algorithms for distributed Web servers", *Proc. ICDCS'97*, Baltimore, MD, May 1997, pp. 169–176.

[5] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, "A scalable and highly available Web server", *Proc. 41st IEEE Computer Society Intl. Conf.* (COMPCON 1996), Feb. 1996, pp. 85-92.

[6] E.D. Katz, M. Butler, R. McGrath, "A scalable HTTP server: the NCSA prototype", *Computer Networks and ISDN Systems*, v. 27, 1994, pp. 155-164.

[7] D.A. Menascé, D. Saha, S.C. da Silva Porto, V.A.F. Álmeida, S.K. Tripathi, "Static and dynamic processor scheduling disciplines in heterogeneous parallel architecture", *J. of Parallel and Distributed Computing*, v. 28, 1995, pp. 1-18.

[8] G.K. Zipf, *Human Behavior and the Principles of Least Effort*, Addison-Wesley, Reading, MA, 1949.