A Performance Study of Robust Load Sharing Strategies for Distributed Heterogeneous Web Server Systems *

Michele Colajanni
Dipartimento di Ingegneria dell'Informazione
Università di Modena
Modena, Italy 41100

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

Abstract

Replication of information across multiple servers is becoming a common approach to support popular Web sites. A distributed architecture with some mechanisms to assign client requests to Web servers is more scalable than any centralized or mirrored architecture. In this paper, we consider distributed systems in which the *Authoritative Domain Name Server* (ADNS) of the Web site takes the request dispatcher role by mapping the URL hostname into the IP address of a visible node that is, a Web server or a Web cluster interface. This architecture can support local and geographical distribution of the Web servers. However, the ADNS controls only a very small fraction of the requests reaching the Web site because the address mapping is not requested for each client access. Indeed, to reduce Internet traffic, address resolution is cached at various name servers for a *time-to-live* (TTL) period. This opens an entirely new set of problems that traditional centralized schedulers of parallel/distributed systems do not have to face. The heterogeneity assumption on Web node capacity, which is much more likely in practice, increases the order of complexity of the request assignment problem,

and severely affects the applicability and performance of the existing load sharing algorithms. We propose new assignment strategies, namely *adaptive TTL* schemes, which tailor the TTL value for each address mapping, instead of using a fixed value for all mapping requests. The *adaptive TTL* schemes are able to address both the non-uniformity of client requests and the heterogeneous capacity of Web server nodes. Extensive simulations show that the proposed algorithms are very effective in avoiding node overload even for high levels of heterogeneity and limited ADNS control.

Index Terms

- Load balancing
- Performance analysis
- Domain Name System
- Distributed systems
- Global scheduling algorithms
- World Wide Web

Author affiliation

Michele Colajanni
Dipartimento di Ingegneria dell'Informazione
University of Modena
Modena, Italy 41100

Philip S. Yu
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598

1 Introduction

With ever increasing traffic demand on popular Web sites, a parallel or distributed architecture can provide transparent access to the users to preserve one logical interface, such as www.site.org. System scalability and transparency require some internal mechanism that automatically assigns client requests to the Web node that can offer the best service [25, 17, 13, 6]. Besides performance improvement, dynamic request assignment allows the Web server system to continue to provide information even after temporary or permanent failures of some of the servers.

In this paper, we consider the Web server system as a collection of heterogeneous nodes, each of them with a visible IP address. One *node* may consist of a single Web server or multiple server machines behind the same network interface as in Web cluster systems. The client request assignment decision is typically taken at the network *Web switch* level when the client request reaches the Web site or at the *Domain Name Server* system (DNS) level during the address lookup phase that is, when the URL hostname of the Web site is translated into the IP address of one of the nodes of the Web server system [8]. Address mapping request is handled by the Authoritative Domain Name Server (ADNS) of the Web site that can therefore serve as request dispatcher. Original installations of locally distributed Web server systems with DNS-based request assignment include NCSA HTTP-server [25], SWEB server [4], lbnamed [34], SunSCALR [36]. On a geographical scale, the Web site is typically built upon a set of distributed Web clusters, where each cluster provides one IP visible address to client applications. DNS-based dispatching mechanisms for geographically distributed systems are implemented by several commercial products such as Cisco DistributedDirector [11], Alteon WebSystems GSLB [3], Resonate's Global Dispatcher [33], F5 Networks 3DNS [19], HydraWeb Techs [23], Radware WSD [31], IBM Network Dispatcher [26], Foundry Networks [21].

For the case of multiple Web servers at the same location (Web clusters), various Web switch solutions are described in [12, 17, 22, 35, 29]. The Web switch has a full control on the incoming requests. However, this approach is best suitable to a *locally* distributed Web server system. Moreover, the Web switch can become the system bottleneck if the Web site is subject to high request rates and there is one dispatching mechanism.

In this paper, we focus on DNS-based architectures that can scale from *locally* to *geographically* distributed Web server systems. The dispatching algorithms implemented at the ADNS level have to address new challenging issues. The main problems for load sharing come from the highly uneven distribution of the load among the client domains [5, 16] and from Internet mechanisms for address caching that let the ADNS control only a very small

fraction, often on the order of a few percentage, of the requests reaching the Web site. The ADNS specifies the period of validity of cached addresses. This value is referred to as the *time-to-live* (TTL) interval and typically fixed to the same time for all address mapping requests reaching the ADNS. Unlike the Web switch that has to manage all client requests reaching the site, the limited control of ADNS prevents risks of bottleneck in DNS-based distributed Web server systems. On the other hand, this feature creates a challenge to ADNS-based *global scheduling*[1] algorithms and makes this subject quite different from existing literature on centralized schedulers of traditional parallel/distributed systems that have almost full control on the job requests [18, 10, 32, 24]. The general view of this problem is to find a mechanisms and algorithms that are able to stabilize the load in a system when the control on arrivals is limited to a few percentage of the total load reaching the system.

Under realistic scenarios, in [13] it is shown that the application of classical dispatching algorithms, such as *round-robin* and *least-loaded-server*, to the ADNS often results in overloaded Web nodes well before the saturation of the overall system capacity. Other dispatching policies that integrate some client information with feedback alarms from highly loaded Web nodes achieve much better performance in a *homogeneous* Web server system.

The complexity of the ADNS assignment problem increases in the presence of Web nodes with different capacities. Web systems with so called *heterogeneous* nodes are much more likely to be found in practice. As a result of non-uniform distribution of the client requests rates, limited control of the dispatcher and node heterogeneity, the ADNS has to take global scheduling decisions under great uncertainties. This paper finds that a simple extension of the algorithms for homogeneous Web server systems proposed in [13] does not perform well. These policies show poor performance even at low levels of node heterogeneity. Other static and dynamic global scheduling policies for heterogeneous parallel systems which are proposed in [2, 27] cannot be used because of the peculiarities of the ADNS scheduling problem.

These qualitative observations and preliminary performance results convinced us that an entirely new approach was necessary. In this paper, we propose and evaluate new ADNS dispatching policies, called *adaptive TTL* algorithms. Unlike conventional ADNS algorithms where a fixed TTL value is used for all address mapping requests, tailoring the TTL value adaptively for each address request opens up a new dimension to perform load sharing. Extensive simulation results show that these strategies are able to avoid overloading nodes very effectively even for high levels of node heterogeneity. Adaptive TTL dispatching is a simple mechanism that can be immediately used in an actual environment because it requires no changes to existing Web protocols and applications, or other parts that are not under the direct control of the Web site technical management. Because the installed base of hosts,

[1] In this paper we use the definition of *global scheduling* given in [10], and *dispatching* as its synonymous.

name servers, and user software is huge, we think that a realistic dispatching mechanism must work without requiring modifications to existing protocols, address mechanisms, and widely used network applications. Moreover, adaptive TTL algorithms have low computational complexity, require a small amount of system information, and show robust performance even in the presence of non-cooperative name servers and when information about the system state is partial or inaccurate.

The outline of the paper is as follows. In Section 2, we provide a general description of the environment. In Section 3, we focus on the new issues that a distributed Web server system introduces on the ADNS global scheduling problem. We further examine the relevant state information required to facilitate ADNS scheduling. In Section 4, we consider various ADNS scheduling algorithms with constant TTL, while we propose the new class of *adaptive TTL* algorithms in Section 5. In Section 6, we describe the model and parameters of the heterogeneous distributed Web server systems for the performance study. Moreover, we discuss the appropriate metrics to compare the performance of the algorithms. In Section 7, we present the performance results of the various algorithms for a wide set of scenarios. In Section 8, we analyze the implications of the performance study and summarize the characteristics of all proposed algorithms. Section 9 contains our concluding remarks.

2 Environment

In this paper, we consider the Web server system as a collection of $\{S_1, \ldots, S_N\}$ heterogeneous nodes that are numbered in non-increasing order of processing capacity. Each node may consist of a single server or a Web cluster, it may be based on single- or multi-processors machines, have different disk speeds and various internal architectures. However, from our point of view, we only address heterogeneity through the notion that each node may have a different capacity to satisfy client requests. In particular, each node S_i is characterized by an *absolute capacity* C_i that is expressed as hits per second it can satisfy, and a *relative capacity* ξ_i which is the ratio between its capacity and the capacity of the most powerful node in the Web server system that is, $\xi_i = C_i/C_1$. Moreover, we measure the *heterogeneity level* of the distributed architecture by the maximum difference between the relative node capacities that is, $\alpha = \xi_1 - \xi_N$. For example, if the absolute capacities of the nodes are C_1=150 hits/sec, C_2=120 hits/sec, C_3=100 hits/sec, C_4=75 hits/sec, the vector of relative capacities is $\xi = \{1, 0.8, 0.66, 0.5\}$, and $\alpha = 0.5$.

The Web site built upon this distributed Web server system is visible to users through one logical hostname, such as www.site.org. However, IP addresses of the Web nodes (that is, servers or clusters) are visible to client applications.

In operation, the WWW works as a client/server system where clients submit requests for objects identified through the Uniform Resource Locator (URL) specified by the user. Each URL consists of a hostname part and a document specification. The hostname is a logical address that may refer to one or multiple IP addresses. In this paper, we consider this latter instance, where each IP address is that of a node of the Web server system. The hostname has to be resolved through an *address mapping request* that is managed by the Domain Name Server System. The so called *lookup phase* may involve several *name servers* and, in some instances, also the ADNS. Once the client has received the IP address of a Web node, it can direct the *document request* to the selected node. The problem is that only a small percentage of client requests actually needs the ADNS to handle the address request. Indeed, on the path from clients to the ADNS, there are typically several name servers, which can have a valid copy of the address mapping returned by the ADNS. When this mapping is found in one of the name servers on this path and the TTL is not expired, the address request is resolved bypassing the name resolution provided by the ADNS. Further, Web browsers at the user side also cache some of the address mapping for a period of usually 15 minutes that is out the ADNS control.

The clients have a (set of) local name server(s) and are connected to the network through local gateways such as firewalls or SOCKS servers. We will refer to the sub-network behind these local gateways as *domain*.

3 DNS-based Web server system

In this section we first consider the various issues on DNS-based dispatching. We then examine the state and configuration information that can be useful to ADNS and discuss ways to obtain this information.

3.1 DNS global scheduling issues

The distributed Web server system uses one URL hostname to provide a single interface for users. For each address request reaching the system, the ADNS returns a tuple (IP address, TTL), where the first tuple entry is the IP address of one of the nodes in the Web server system, and the second entry is the TTL period during which the name servers along the path from the ADNS to the client cache the mapping. In addition to the role of IP address resolver, the ADNS of a distributed Web server system can perform as a global scheduler that distributes the requests based on some optimization criterion, such as load balancing, minimization of the system response time, minimization of overloaded nodes,

client proximity.

We will first consider the scheduling issue on address mapping and delay the discussion on TTL value selection until Section 5 which is the new approach introduced in this paper.

Existing ADNSes typically use DNS rotation or Round-Robin (RR) [1], least-loaded-server [34] or proximity [11] algorithms to map requests to the nodes. We observed that these policies show fine performance under (unrealistic) hypotheses that is, the ADNS has almost full control on client requests and the clients are uniformly distributed among the domains. Unfortunately, in the Web environment, the distribution of clients among the domains is highly non-uniform [5] and the issue of the limited ADNS control cannot be easily removed. Indeed, IP address caching at name servers for the TTL period limits the control of the ADNS to a small fraction of the requests reaching the Web server system. Although TTL values close to 0 would give more control to the ADNS, various reasons prevent this solution. Besides the risks of causing bottleneck at the ADNS, very small TTL values are typically ignored by the name servers in order to avoid overloading the network with name resolution traffic. The ADNS assignment is a very coarse grain distribution of the load among the Web nodes, because proximity does not take into account heavy load fluctuations of Web workload that are amplified by the geographical context. Besides burst arrivals and not uniform distribution among Internet domains of clients connected to the Web site, world time zones are another cause of heterogeneous source arrivals. As we are considering highly popular Web sites, if the ADNS selects the Web node only on the basis of the best network proximity, it is highly probable that address resolution is found in the caches of the intermediate name servers of the Internet Region, so that even less address requests will reach the ADNS.

From the Web server system point of view, the combination of non-uniform load and limited control can result in bursts of requests arriving from a domain to the same node during the TTL period, thereby causing high load imbalance. The main challenge is to find a realistic ADNS algorithm, with low computational complexity and fully compatible with Web standards, that is able to address these issues and the heterogeneity of the Web nodes.

3.2 Relevant state and configuration information for DNS scheduling

One important consideration in dealing with the ADNS scheduling problem is the kind of state and configuration information that can be used in mapping URL host names to IP addresses. An in-depth analysis carried out in [13] on the effectiveness of the different types of state information on ADNS scheduling for homogeneous Web server systems is summarized

below:

No state information. Scheduling policies, such as round-robin and random used in [25, 4, 28], that do not require any state information show very bad performance under realistic scenarios [13]. (See further discussions in Section 7.1.)

Detailed server state information. Algorithms using detailed information about the state of each Web node (for example, queue lengths, present and past utilization) perform better than the previous ones, but are still unable to avoid overloading some Web node while under-utilizing other nodes. The present load information does not capture the TTL effect that is, the future arrivals due to past address resolutions. This makes the server load information obsolete quickly and poorly correlated with future load conditions. This excludes policies of the least-loaded-server class from further consideration in a heterogeneous Web server system.

Information on client domain load. An effective scheduling policy has to take into account some client domain information, because any ADNS decision on an IP address resolution affects the selected node for the entire TTL interval during which the hostname to IP address mapping is cached in the name servers. Therefore, the ADNS needs to make an adequate prediction about the impact on the future load of the nodes following each address mapping. The key goal is to obtain an estimation of the ***domain hit rate***, λ_i, which is the number of hits per second reaching the Web server system from the i-th domain. Multiplying λ_i by TTL, we obtain the ***hidden load weight*** which is the average number of hits that each domain sends to a Web node during a TTL interval after a new address resolution request has reached the ADNS.

Information on overloaded nodes. Information on overload nodes is useful so that ADNS can avoid assigning address requests to already over-utilized nodes. For the purpose of excluding them from any assignment until their load returns in normal conditions, scheduling algorithms can combine the domain hit rate information with some feedback information from the overloaded nodes to make address mapping decision.

In addition,

Node processing capacities. As we shall see later, in a heterogeneous Web server system, the node processing capacity needs to be taken into account by ADNS in either making node assignment or fixing the TTL value.

3.3 Information gathering mechanisms

Based on the observation from the previous section, all ADNS scheduling algorithms considered in this paper will apply the feedback alarm mechanism and evaluate the hit rate of each client domain connected to the Web site. The main question is whether these kinds of information are actually accessible to the ADNS of a distributed Web server system. We recall that the first requirement for an ADNS algorithm is that it must be fully compatible with existing Web standards and protocols. In particular, all state information needed by a policy has to be received by the ADNS from the nodes and the ADNS itself, because they are the only entities that the Web site management can use to collect and exchange load information. Algorithms and mechanisms that need some active cooperation from any other Web components, such as browsers, name servers, users, will not be pursued because they require modifications of some out-of-control Web components. We next examine how the ADNS can have access to the feedback alarm and the domain hit rate information.

The implementation of the *feedback alarm* information requires two simple mechanisms: a monitor of the load of each Web node, and an asynchronous communication protocol between the nodes and the ADNS. Each node periodically calculates its utilization and checks whether it has exceeded a given ϑ threshold. In that case, the node sends an *alarm signal* to the ADNS that excludes it from any further assignment until its load falls below the threshold. This last event is communicated to the ADNS through a *normal signal*. We assume that all of the global scheduling algorithms to be discussed next consider a node as a candidate for receiving requests only if that node is not overloaded. Although fault-tolerance is not the focus of this paper, it is worth noting that a very simple modification of this feedback mechanism could also avoid routing requests to failed or unreachable nodes. Either node-initiated (through synchronous messages) or scheduler-initiated (through a polling mechanism) strategies could be combined with the same scheduling algorithms above to also provide fault-tolerance.

The estimation of the domain hit rate cannot be done by the ADNS alone because the information coming from the clients to the ADNS is very limited. For each new session requiring an address resolution, the ADNS sees only the IP address of a client's domain. Due to the address caching mechanisms, the ADNS will see another address request coming from the same domain only after TTL seconds, independent of the domain hit rate. Hence, the only viable approach to estimate this information requires cooperation of the Web nodes. They can track and collect the workload to the distributed Web server system through the *logfile* maintained by each node to trace the client accesses in terms of hits. According to the Common Logfile Format [14], the information 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 (additional details can be found in [9]). Each node periodically sends its estimate of the domain hit rates to the ADNS, where a collector process gets all estimates and computes the actual hit rate from each domain by adding up its hit rate on each node.

Finally, the node processing capacity information ($\{\xi_i\}$) required is a static configuration information. It is an estimate on the number of hits or HTTP requests per second each node can support.

Figure 1 summarizes the various components needed for ADNS assuming that a node consists of a single Web server machine. In addition to the DNS base function, these include an ADNS scheduler, alarm monitor, domain load collector and TTL selector. The ADNS scheduler assigns each address request to one of the node based on some scheduling algorithm. The alarm monitor tracks the feedback alarm from servers to avoid assigning requests to an overloaded node until the load level is returned to normal, while the domain load collector collects the domain hit information from each node and estimates the hit rate and hidden load weight of each domain. The TTL selector fixes the appropriate TTL value for the address mapping. Also shown is the corresponding components in the Web node. Besides the HTTP daemon server, these include the load monitor and request counter. The load monitor tracks the node load and issues alarm and normal signal accordingly as explained above. The request counter estimates the number of hits received from each domain in a given period and provides the information to the domain load collector in ADNS. When the node is a Web cluster with multiple server machines, the request counter and load monitor processes run on the Web switch. This component would have the twofold role of intra-cluster information collector and interface with the ADNS.

4 DNS algorithms with constant TTL

Strategies that do not work well in the homogeneous case cannot be expected to achieve acceptable results in a heterogeneous node system. Hence, we consider only the better performing homogeneous node algorithms that seem to be extensible to an heterogeneous environment. Among the several alternatives proposed in [13], the following policies gave the most promising results. We present them in the forms extended to a heterogeneous node system to take into account both the non-uniform hit rates and different node capacities.

Two-tier Round-Robin (RR2). This algorithm is a generalization of the Round-Robin (RR) algorithm. It is based on two considerations. First of all, since the clients are

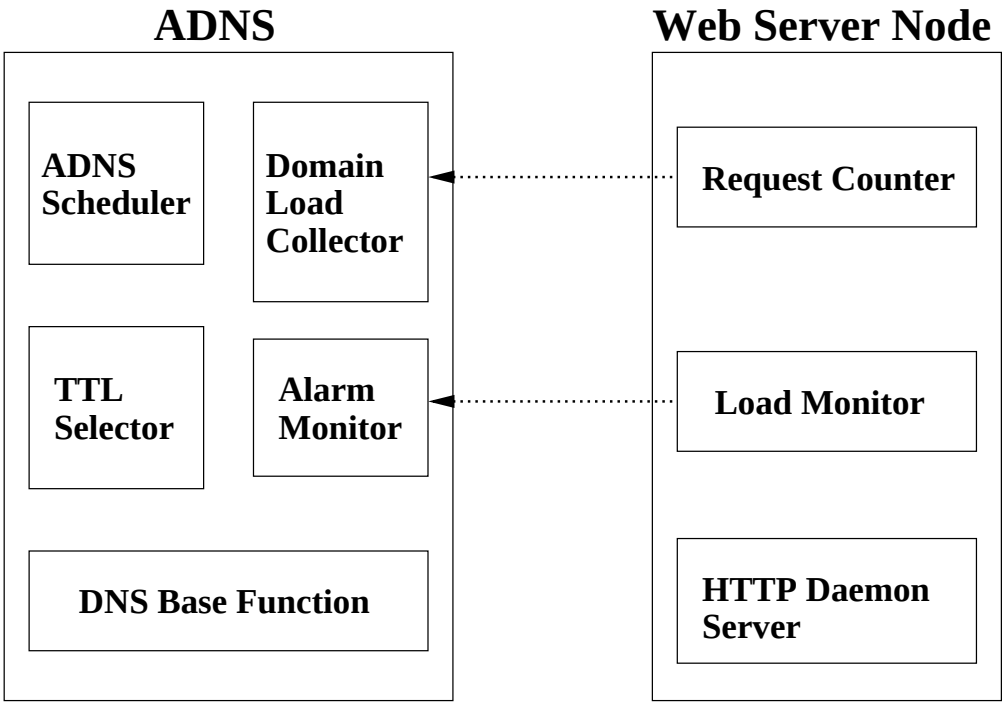

Figure 1: Software component diagram

unevenly distributed, the domain hit rates are very different. Secondly, the risk of overloading some of the nodes is typically due to the requests coming from a small set of very popular domains. Therefore, RR2 uses the domain hit rate information to partition the domains connected to the Web site into two classes: *normal* and *hot* domains. In particular, RR2 sets a *class threshold* and evaluates the *relative domain hit rate*, which is with respect to the total number of hits in an interval from all connected domains. The domains characterized by a relative hit rate larger than the class threshold belong to the hot class. By default, we fix the class threshold to $1/|D|$, where $|D|$ is the average number of domains connected to the nodes. That is to say, each domain with a relative hit rate larger than the class threshold belongs to the hot class. 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 nodes. Partitions of domains in more than two classes have been investigated with little performance improvement [13].

RR2 (and also RR) is easily extendible to a heterogeneous Web server system through the addition of some *probabilistic* routing features. The basic idea is to make the round robin assignment probabilistic based upon the node capacity. To this purpose, we generate a random number ϱ ($0 \leq \varrho \leq 1$) and, assuming that S_{i-1} was the last chosen node, we assign the new requests to S_i only if $\varrho \leq \xi_i$. Otherwise, S_{i+1} becomes the next candidate and we repeat the process that is, we generate another random number and compare it with the relative capacity of S_{i+1}. This straightforward modification allows RR2 and RR to schedule the requests by taking into account of the various node capacities. These probabilistic versions of the RR and RR2 algorithms are denoted by *Probabilistic-RR* (PRR) and *Probabilistic-RR2* (PRR2), respectively. Hereafter, we will refer to the conventional RR as *Deterministic-RR* (DRR) to distinguish it from PRR and analogously, DRR2 from PRR2.

Dynamically Accumulated Load (DAL). This algorithm uses the domain hit rate to estimate the *hidden load weight* of each domain. Each time the ADNS makes a node selection following an IP address resolution request, it accumulates the hidden load weight of the requesting domain in a bin for each node to predict how many requests will arrive to the chosen node due to this mapping. At each new IP address request, the ADNS selects the node that has the lowest accumulated bin level.

DAL makes the node selection only based on the hidden load weight from the clients. A generalization of this algorithm to a heterogeneous Web server system should take into account the node capacity. The solution is to normalize the hidden load weight accumulated at each bin by the capacity of the corresponding node. On IP address

assignment, DAL now selects the node that would result in the lowest bin level after the assignment.

Minimum Residual Load (MRL). This algorithm is a modification of the basic DAL. Analogous to the previous algorithm, MRL tracks the hidden load weight of each domain. In addition, the ADNS maintains an *assignment table* containing all domain to node assignments and their times of occurrences. Let l_j be the average session length of a client of the j-th domain. After a period of TTL+l_j, the effect of the assignment is expected to expire that is, no more requests will be sent from the j-th domain due to this assignment. Hence, the entry for that assignment can be deleted from the assignment table. At the arrival of an address resolution request at the time t_{now}, the ADNS evaluates the expected number of *residual* requests that each node should have, on the basis of the previous assignments, and chooses the node with the minimum number of residual requests that is,

$$\min_{i=1,\ldots,N} \left\{ \sum_{domain_j \rightarrow node_i} \sum_{k} \left[(w_j/\xi_i) \left(t_j(i,k) + TTL + l_j - t_{now} \right)_+ \right] / (TTL + l_j) \right\} \quad (1)$$

where w_j is the hidden load weight of the j-th domain, ξ_i is the relative capacity of the i-th node, and $t_j(i,k)$ is the time of the assignment of the $k-$th address resolution request coming from the j-th domain to the i-th node in the mapping table. The $(x)_+$ notation denotes that only the positive terms are considered in the internal sum because no more residual load is expected to remain from an assignment when the corresponding term is detected to be negative. The term $(t_j(i,k) + TTL + l_j)$ represents the time instant that the address mapping expires, and the term $(t_j(i,k) + TTL + l_j - t_{now})_+$ represents the remaining time that the mapping is still valid. By normalizing w_j by ξ_i, the effect of the node heterogeneity is captured. The average session lengths l_j are not readily available at the ADNS but they can be estimated at the Web nodes. A session can be identified via a *cookie generation* mechanism or inferred through some heuristics using the site's topology or referred information [30]. Since l_j is expected to be rather stable over time, the frequency of exchanges between nodes and ADNS for this information should be relatively low.

In addition to the extensions of the ADNS algorithms taken from homogeneous environments, we now consider some scheduling disciplines that are specifically tailored to a heterogeneous environment. The basic idea is to reduce the probability of assigning requests from hot domains to less powerful nodes. Two representative examples of this approach are:

Two-alarm algorithms. This strategy modifies the asynchronous feedback mechanism by introducing two-levels of threshold (ϑ_1,ϑ_2) with a different relative capacity corresponding to each level. The goal is to reduce the probability that a node with high load gets selected to serve requests. Since the strategies PRR and PRR2 base their decision on the relative node capacities, we force a reduction of the *perceived* relative capacity, as node utilization becomes high. In fact, when the utilization of a node S_i exceeds the first threshold ϑ_1, the ADNS reduces its relative capacity, for example by dividing ξ_i by two. When the utilization exceeds the second threshold ϑ_2, the ADNS fixes ξ_i to 0. If all capacities were 0, we would use a random choice weighted on the node capacities.

Restricted-RR2. This algorithm is a simple modification of RR2. The requests coming from the *normal* domains are divided among all the nodes in the same probabilistic manner as previously described, while the requests coming from the *hot* domains are assigned only to the top (that is, more powerful) nodes of the distributed Web server system. To avoid having too restricted a subset to serve the heavy requests, we consider as a *top node* any node with a relative capacity (ξ_i) of 0.8 or more.

5 DNS algorithms with dynamic choice of TTL

As we shall see in Section 7.1, the algorithms derived as generalizations of those scheduling policies used in a homogeneous Web server system are inadequate to address node heterogeneity, unevenly domain hit rates, and limited ADNS control. Their poor performance motivated the search for new strategies that intervene on the TTL value, which is the other parameter controlled by the ADNS. In this section, we propose two classes of algorithms that use some policy of Section 4 for the selection of the node and dynamically adjust the TTL value based on different criteria. The first class of algorithms mainly addresses the problem of the limited control of the ADNS. To this purpose, it increases the ADNS control when there are many overloaded nodes through a dynamic reduction of the TTL. The second class of algorithms is more oriented to address node heterogeneity and unevenly distributed clients through the use of TTL values that reduce the load skew. A summary of main characteristics of all discussed dispatching algorithms is in Table 2, Section 8.

As we shall see, to set the TTL value for each address request, the variable TTL algorithm requires information on the number of overloaded nodes, while adaptive TTL algorithms need information on the processing capacity of each node and the hit rate of each connected domain. Since these are the same type of information used by the ADNS scheduler discussed in the previous section, no new information is required to dynamically fix the TTL value.

5.1 Variable TTL algorithms

Firstly, we consider the *variable TTL* (varTTL) algorithms. They tune TTL values based on the load conditions of the overall Web server system. When the number of overloaded nodes increases, these algorithms reduce the TTL value. Otherwise, they use the default or higher TTL values. The rationale for these strategies comes from the principle that the ADNS should have more control on the incoming requests when many of the nodes in the distributed system are overloaded.

In this paper, we implement the following simple formula for determining the TTL value at time t,

$$TTL(t) = TTL_{base} - \psi * (|overload(t)|) \tag{2}$$

where TTL_{base} is the TTL value when no node is overloaded and $|overload(t)|$ is the minimum between some upper bound value (ω) and the number of overloaded nodes at time t. For example, if TTL_{base} is 300 seconds and ψ is 60 seconds, the TTL value drops to 240 seconds after one node gets overloaded. By fixing ω to 4, it means that the TTL value can only drops to 60 seconds, even if there are more than 4 nodes overloaded.

For the node selection, the varTTL policies can be combined with any algorithm of Section 4. In this paper, we consider the probabilistic versions of RR and RR2 that is, PRR-varTTL and PRR2-varTTL.

5.2 Adaptive TTL algorithms

Instead of just reducing the TTL value to give more control to the ADNS, an alternative is to address the unevenly distributed hit rates or heterogeneous node capacities by assigning a different TTL value to each address request. The rationale for this approach comes from the observation that the hidden load weight increases with the TTL value, independently of the domain. Therefore, by properly selecting the TTL value for each address resolution request, we can control the subsequent request load to reduce the load skews that are the main cause of overloading, especially in a heterogeneous system. More specifically, we can make the subsequent requests from each domain to consume similar percentages of node capacity. This can address both node heterogeneity and non-uniform hit rates.

First consider node heterogeneity. We assign a higher TTL value when the ADNS chooses a more powerful node, and a lower TTL value when the requests are routed to a less capable node. This is due to the fact that for the same fraction of node capacity, the more powerful node can handle a larger number of requests, or take requests for a longer TTL interval.

An analogous approach can be adopted to handle the uneven hit rate distribution. The address requests coming from hot domains will receive a lower TTL value than the requests originated by normal domains. As the hot domains have higher hit rates, a shorter TTL interval will even out the total number of subsequent requests generated.

The new class of scheduling disciplines that use this approach is called *adaptive TTL*. It consists of a two-step decision process. In the first step, the ADNS selects the Web node. In the second step, it chooses the appropriate value for the TTL interval. These strategies can be combined with any scheduling algorithm described in Section 4. Due to space limitations, we only consider the basic RR algorithm and its RR2 variant. Furthermore, we combine the adaptive TTL policies with the *deterministic* and *probabilistic* versions of these algorithms. Both of them handle non-uniform requests by using TTL values inversely proportional to the domain hit rate, while address system heterogeneity either during the node selection (*probabilistic* policies) or through the use of TTL values proportional to the node capacities (*deterministic* policies).

5.2.1 Probabilistic algorithms

The *probabilistic* policies use PRR or PRR2 algorithms to select the node. After that, the TTL value is assigned based on the hit rate of the domain that has originated the address request. In its most generic form, we denote by TTL/i the policy that partitions the domains into i classes based on the *relative domain hit rate* and assigns a different TTL value to address requests originating 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 a truly adaptive TTL algorithm. For $i = 2$, we have the policy (TTL/2) that partitions the domains into *normal* and *hot* domains, and chooses a high TTL value for requests coming from normal domains, and a low TTL value 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 value for each connected domain. (In actual implementation, to reduce the amount of bookkeeping, domains with lower hit rates may be lumped into one class.) For TTL/K policies, let $TTL_j(t)$ denote the TTL value chosen for the requests coming from the j-th domain at time t,

$$TTL_j(t) = \frac{\lambda_{max}(t)\eta_p}{\lambda_j(t)} \tag{3}$$

where η_p is the parameter which scales the average TTL (and the minimum TTL) value and hence overall rate of the address mapping requests, $\lambda_j(t)$ and $\lambda_{max}(t)$ are the hit rates

(available as an estimation at time t) of the j-th domain and the most popular domain, respectively.

5.2.2 Deterministic algorithms

Under the deterministic algorithms, the node selection is done by the ADNS through the deterministic RR or RR2 policy. The approach for handling non-uniform hit rates is via adjusting TTL value similar to that described for the probabilistic disciplines. However, the TTL value is now chosen by considering the node capacity as well. For the generic TTL/S_i policy, we partition the client domains into i classes based on the domain hit rates. The TTL for each class and node is set inversely proportional to the class hit rate while proportional to the node capacity. The deterministic TTL/S_1 algorithm is a degenerate case that considers node heterogeneity only and ignores the skew on domain hit rates.

The TTL/S_2 policy uses two TTL values for each node depending on the domain class of the requests that is, normal or hot domain.

The TTL/S_K algorithm selects a TTL value for each node and domain combination. Specifically, let $TTL_{ij}(t)$ be the TTL chosen for the requests from the j-th domain to the i-th node at time t,

$$TTL_{ij}(t) = \frac{\lambda_{max}(t)\eta_d\xi_i}{\lambda_j(t)} \tag{4}$$

where η_d is the parameter which scales the average TTL (and the minimum TTL) value.

6 Performance model

In this section we provide details on the simulation model and describe the various parameters of the model. We then discuss the performance metrics to compare the different ADNS schemes.

6.1 Model assumptions and parameters

We first consider the workload. 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 $1/i^{(1-x)}$ [38]. This choice is motivated by several studies demonstrating that if one ranks the popularity of client domains by the frequency of their accesses to the Web site, the distribution of 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 [5]. In the experiments, the clients are partitioned among the domains based on a pure Zipf's distribution that is, using $x = 0$, in the default case. This represents the most uneven client distribution. Additional sensitivity analysis on the skew parameter (x) and other distributions is included in Section 7.4.

We did not model the details of Internet traffic [15] because the focus of this paper is on Web server system. However, we consider major components that impact the performance of the system. This includes an accurate representation of the number and distribution of the intermediate name servers as in [7], because they affect operations and performance of the ADNS scheduling algorithms through their address caching mechanisms.

Moreover, we consider all the details concerning a client *session* that is the entire period of access to the Web site from a single user. In the first step, the client obtains (through the ADNS or the cache of a name server or gateway) an address mapping to one of the Web nodes through the address resolution process. As the Web server system consists of heterogeneous nodes with identical content, the requests from the clients can be assigned to any one of the nodes. Once the node selection has been completed, the client is modeled to submit multiple Web page requests that are separated with a given mean *think time*. The number of page requests per session and the time between two page requests from the same client are assumed to be exponentially distributed as in [5].

Each page request consists of a burst of small requests sent to the node. These bursts represent the objects that are contained within a Web page. These are referred to as *hits*. Under the HTTP/1.0 protocol, each hit request establishes a new connection between the client and the Web node. However, address caching at the browser level guarantees that a client session is served by the same Web node independently of its duration. Moreover, the new version HTTP/1.1 provides persistent connections during the same session [20]. As a consequence, the difference between HTTP protocols does not affect the results of this paper.

The number of hits per page request are obtained from a uniform distribution in the discrete interval [5-15]. Previous measures reported roughly seven different hits per page, however more recent analyses indicate that the mean number of embedded hits is increasing [16]. The hit service time and the inter-arrival time of hit requests to the node are assumed to be exponentially distributed. (We will also consider the case of hits with very long mean service time like the CGI type in Section 7.4.) Other parameters used in the experiments are reported in Table 1 with their default values between brackets. When not otherwise

specified, all performance results refer to the default values. A thorough sensitivity analysis, not shown because of space limits, reveals that main conclusions of the experiments are not affected by the choice of workload parameters such as the number of hits per page request, the mean service time and inter-arrival time of hits.

Category	*Parameter*	*Setting (default values)*
Web system	Number of nodes	7
	System capacity	1500 hits/sec
	Average system load	1000 hits/sec
	Average utilization	0.6667
	Homogeneous $\alpha_A = 0$	ξ_A=[1,1,1,1,1,1,1]
	Heterogeneity $\alpha_B = 0.2$	ξ_B=[1,1,1,0.8,0.8,0.8,0.8]
	Heterogeneity $\alpha_C = 0.35$	ξ_C=[1,1,0.8,0.8,0.65,0.65,0.65]
	Heterogeneity $\alpha_D = 0.5$	ξ_D=[1,1,0.8,0.8,0.5,0.5,0.5]
	Heterogeneity $\alpha_E = 0.65$	ξ_E=[1,1,0.8,0.8,0.35,0.35,0.35]
Domain	Connected	10-100 (20)
	TTL (constant)	0-700 (240)
	TTL (adaptive)	[0-2400]
Client	Number	1000-3000 (1500)
	Distribution among domains	Zipf ($x = 0, 0.5, 1$)
		Geometric ($p = 0.3$)
Request	Web page requests per session	exponential (mean 20)
	Hits per Web page request	uniform in [5-15]
	Inter-arrival of page requests	exponential (mean 15)
	Inter-arrival of hits	exponential (mean 0.25)
	Hit service time	exponential (mean $1/C_i$)

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

In our experiments, we considered five levels of node heterogeneity. Table 1 reports details about the *relative capacities* and the *heterogeneity level* of each Web server system. By carefully choosing the workload and system parameters, the average utilization of the system is kept to 2/3 of the whole capacity. This value is obtained as a ratio between the *offered load* that is, the total number of hits per second arriving to the Web site, and the *system capacity* which is the sum of the capacity of each node denoted in hits per second. Although we considered different levels of node heterogeneity, we keep the system capacity constant to allow for a fair comparison among the performance of the proposed algorithms.

Furthermore, to implement the feedback alarm, each node periodically calculates its utilization (the period is of 16 seconds) and checks whether it has exceeded a given ϑ=0.75 threshold as in [13]. (Additional simulation results, not shown, indicate that our results are not sensitive to these parameters.) For two-alarm algorithms, the two threshold are fixed at

0.6 and 0.75, respectively.

For the constant TTL schemes, a default TTL value of 240 seconds is used as in [17]. For the variable TTL algorithm, TTL_{base} is fixed to 300 seconds with $\omega = 4$. We note that the base TTL value (TTL_{base}) is higher than the TTL value in the constant TTL case. In fact, if one of the nodes becomes overloaded, the TTL value drops to 240 seconds which becomes the same as the constant TTL case. For the adaptive TTL algorithms, the average TTL value is fixed to 400 seconds so as to keep the minimum TTL value (denoted as $\overline{TTL}$) above 60 seconds, while the maximum TTL value can go up to 1200 seconds. This average TTL value is considerable higher than the 240 seconds in the constant TTL case. Nonetheless, as we shall see later, the adaptive TTL algorithms still perform far better than the constant TTL algorithms. Sensitivity analysis to TTL values is provided in Section 7.4.

The simulators were implemented using the CSIM package [37]. Each simulation run is made up of five hours of the Web site activities. Confidence intervals were estimated, and the 95% confidence interval was observed to be within 4% of the mean.

6.2 Performance metrics

We next examine the metrics of interest for evaluating the performance of an ADNS algorithm in a heterogeneous Web server system. The main goal is to avoid any of the Web nodes becoming overloaded. That is to say, our objective is to minimize the highest load among all nodes at any instant. Commonly adopted metrics such as the standard deviation of node utilization are not useful for this purpose because minimizing the load differences among the Web nodes is only a secondary goal.

These considerations lead us to evaluate the performance of the various policies focusing on the *system maximum utilization* at a given instant that is, the highest node utilization observed at that instant among all nodes in the system. For example, assume three nodes in a Web server system. If their utilizations are 0.6, 0.75, and 0.63, respectively, at time t_1, and 0.93, 0.66 and 0.42, respectively, at time t_2, the system maximum utilization at t_1 is 0.75 and that at t_2 is 0.93. With a system maximum utilization of 0.93, the Web site has serious load problems at t_2.

Specifically, the major performance criterion is the *cumulative frequency* of the system maximum utilization that is, the probability (or fraction of time) that the system maximum utilization is below a certain value. By focusing on the highest utilization among all Web nodes, we can deduce whether the Web server system is overloaded or not. Moreover, its cumulative frequency can provide an indication on the relative frequency of overloading. For

example, if the probability of all nodes less than 0.80 utilized is 0.75, it implies that the probability of at least one node exceeding 0.80 utilized is 0.25.

In practice, the performance of the various scheduling policies is evaluated by tracking at periodic intervals the system maximum utilizations observed during the simulation runs. The node with the maximum utilization changes over time. However, if the system maximum utilization at an instant is low, it means that no node is overloaded at that time. By tracking the period of time the system maximum utilization is above or below a certain threshold, we can get an indication of how well the Web server system is working. We recall that in all experiments the Web server system is subject to an offered load equal to 2/3 of the overall system capacity. We note that typically all the nodes, even if with different proportions, contribute to this maximum during an entire simulation run. Since the average utilization is fixed at 0.6667, the distribution of the system maximum utilization of a perfect policy (always maintaining a utilization of 0.6667 at each node) should be a step function which goes from 0 to 1 at a utilization of 0.6667.

When we evaluate the sensitivity of the algorithms as a function of system parameters, such as node heterogeneity, we find it useful to adopt a different metric that is related to the cumulative frequency of the system maximum utilization. For this set of results, we consider the 96-th percentile of the system maximum utilization that is, $Prob(SystemMaxUtilization < 0.96)$. In other words, the probability that no node of the Web server system is overloaded, namely *Prob(Not Overloaded System)*, becomes the performance metric of interest.

7 Performance results

For the performance evaluation of the proposed dispatching algorithms, we carried out a large number of experiments. Only a subset is presented here due to space limitation. The first set of experiments (in Section 7.1) shows the problem with constant TTL algorithms. It illustrates the point that just considering the scheduling component is not sufficient to achieve good performance. We then evaluate algorithms that also explore the TTL component. The remaining sections focus on measuring how effectively the *adaptive TTL* algorithms applied to a ADNS scheduler, that controls only a small percentage of the requests, can avoid overloading nodes in a heterogeneous distributed Web server system.

7.1 Constant TTL schemes

In this section we evaluate the performance of the constant TTL algorithms based on the parameters shown in Table 1. We obtained simulation results for four heterogeneity levels of the Web server system from 20% to 65%. In Figures 2, we present the performance for the lowest heterogeneity level in Table 1. In this figure, we also report as "ideal" policy the PRR algorithm under uniform distribution of the client request rates, and the random algorithm that has the worst performance. The y-axis is the cumulative probability (or relative frequency) of the system maximum utilization reported on the x-axis. The higher the probability the less likely that some of the nodes will be overloaded, and hence better load sharing is achieved. For example, under the PRR2 policy, the probability of having maximum utilization less than 0.95 is about 0.5, while under DRR, the probability is only 0.2. This figure confirms that the various probabilistic versions of the round-robin policy perform better than the deterministic versions, and this improvement is even more consistent if we look at the RR2 algorithm. The results of the MRL policy are close to PRR2 and much better than the DRR algorithm which is often proposed for DNS-based distributed systems. Indeed, for this latter policy, the probability that no node is overloaded is below 0.2. That is to say, for more than 80% of the observed time, there is at least one overloaded node.

However, even considering this slightly heterogeneous system, no strategy achieves acceptable performance. In the best instance, the Web server system has at least one node overloaded ($Prob(SystemMaxUtilization > 0.96)$) for about 30% of the time, and the shapes of all cumulative frequencies are very far from the "ideal" policy's behavior. This motivated the search for alternative policies such as Restricted-RR2 and Round-Robin with two alarms (Restricted-RR2 and PRR2-Alarm2, respectively).

Figures 3 analyzes the sensitivity of the proposed algorithms with *constant TTL* to the system heterogeneity. Now the y-axis is the probability that no Web node is overloaded, while the x-axis is the heterogeneity level of the Web server system. This figure shows that both DAL and MRL-based policies are unable to control node load when the distributed Web server system is heterogeneous. The performance of the other variants of constant TTL algorithms (that is, PRR-Alarm2, PRR2-Alarm2 and Restricted-RR2) is similar to that of the basic PRR2. Although the PRR2-Alarm2 algorithm often performs better than other policies, no strategy clearly outperforms all the other policies over all system heterogeneity levels. Moreover, the difficulty in determining the best strategy is also confirmed by other (not reported) experiments in which we vary the number of domains, clients and average load. None of these policies can actually be considered adequate because the probability of having at least one overloaded node is still high that is, always more than 0.30.

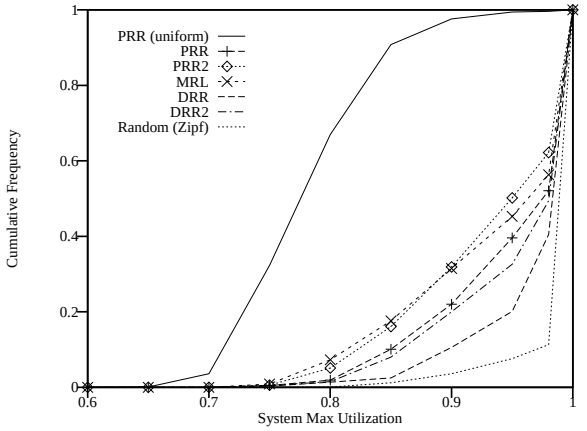

Figure 2: Performance of *constant TTL* algorithms (Heterogeneity level of 20%)

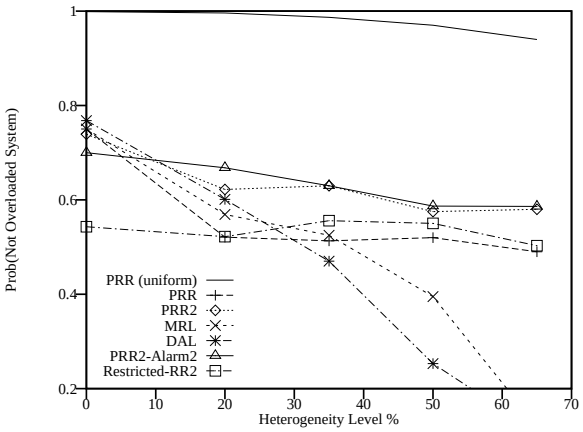

Figure 3: Sensitivity to system heterogeneity of *constant TTL* algorithms

7.2 Comparison of constant and dynamic TTL schemes

The next set of results evaluates how system heterogeneity affects the performance of the *adaptive TTL* schemes. Figure 4 compares various deterministic TTL/S_i algorithms for a low heterogeneity level of 20%. Each set consists of both the deterministic RR2 and RR scheduling schemes for $i = 1, 2$ and K. Also shown in this figure is the DRR scheme with a constant TTL. First of all, for each set of TTL strategies, the RR2 scheduling scheme is always slightly better than the RR scheme. All *adaptive TTL* schemes that address both node and client heterogeneity perform significantly better than *constant TTL* policies, while policies taking into account only the node heterogeneity (as done by TTL/S_1 schemes) do not improve performance much. Moreover, the results of the strategies that use a different TTL for each node and domain, namely DRR-TTL/S_K and DRR2-TTL/S_K, are very close to the envelope curve of the "ideal" PRR(uniform) policy.

Similar results are achieved by the probabilistic schemes that combine adaptive TTL to handle non-uniform domain hit rates and probabilistic routing features to address system heterogeneity. Figure 5 shows the cumulative probability of the system maximum utilization for a heterogeneity level of 20%. The relative order among the strategies remains analogous to the previous order. Specifically, the RR2 scheduling policies are slightly better than RR strategies and TTL/K strategies outperform TTL/2 strategies. Also we consider here the *variable TTL* (varTTL) schemes. The performance of varTTL schemes are close to that of the TTL/2 strategies. Furthermore, all probabilistic *adaptive TTL* approaches are consistently better than the PRR scheme with a *constant TTL*, even for a heterogeneity level low as 20%.

Since the RR2-based algorithms perform better than RR-based counterpart, in the re-

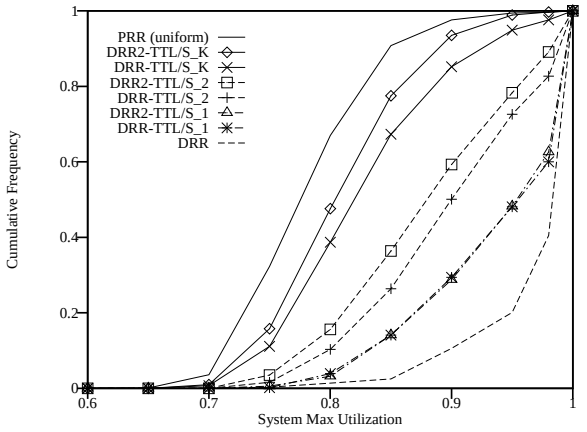

Figure 4: Performance of *deterministic* algorithms (Heterogeneity level of 20%)

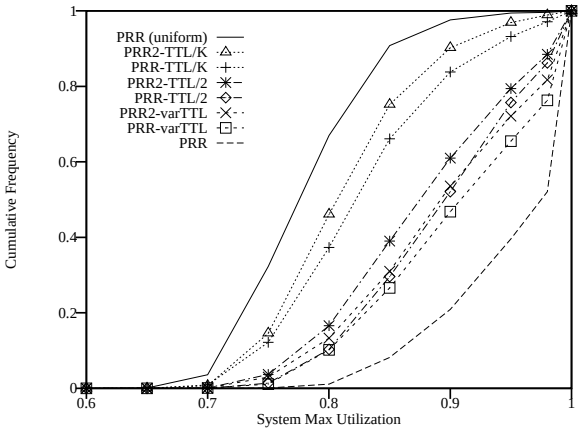

Figure 5: Performance of *probabilistic* algorithms (Heterogeneity level of 20%)

mainder of this section we mainly focus on the former class of policies. Figures 6 and 7 refer to a distributed Web server system with a 35% and 65% heterogeneity level, respectively. Figure 6 shows that when we adopt a TTL proportional to each domain hit rate, the deterministic strategy TTL/S_K prevails over the probabilistic TTL/K. On the other hand, when we consider only two classes of domains, the probabilistic approach TTL/2 is slightly better than the deterministic TTL/S_2. Analogous results are observed for a system heterogeneity equal to 50% and for other system parameters.

The probabilistic approaches tend to perform better than deterministic strategies when the system heterogeneity is very high that is, more than 60%. Figure 7 shows that DRR2-TTL/S_K performs the best if we look at the 98th percentile, while the shape of the curve is in favor of PRR2-TTL/K if we consider lower percentiles. Moreover, the DRR2-TTL/S_2 and PRR2-TTL/2 algorithms, which performed more or less the same in the previous cases, differentiate themselves here in favor of the probabilistic algorithms. PRR2-varTTL performs close to the PRR2-TTL/2 strategy, while DRR2-TTL/S_2 and DRR2-TTL/S_1 seem rather inadequate to address high heterogeneity levels.

We next consider the average number of messages to ADNS under the constant TTL, variable TTL and adaptive TTL schemes. Specifically, in Figure 8, PRR2, PRR2-varTTL and PRR2-TTL/K are chosen to represent the constant TTL, variable TTL and adaptive TTL schemes, respectively. (The difference among the schemes in each class, such as PRR2-TTL/K and DRR2-TTL/S_K, is small.) Figure 8 shows both the number of address requests and alarm requests with a 35% heterogeneity level. The variable TTL schemes have a higher request load to ADNS, while the adaptive TTL and constant TTL schemes are comparable. However, it is important that no policy risks to stress ADNS.

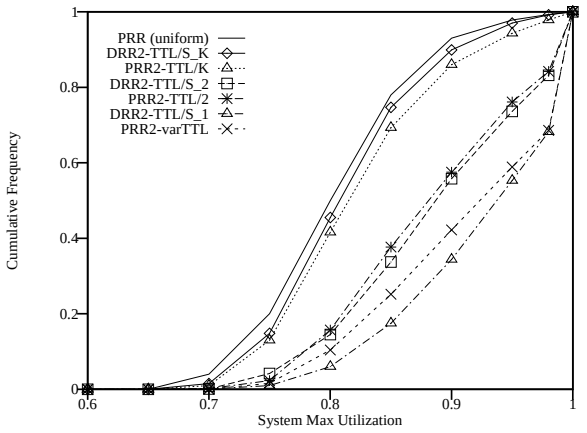

Figure 6: Performance of RR2-based algorithms (Heterogeneity level of 35%)

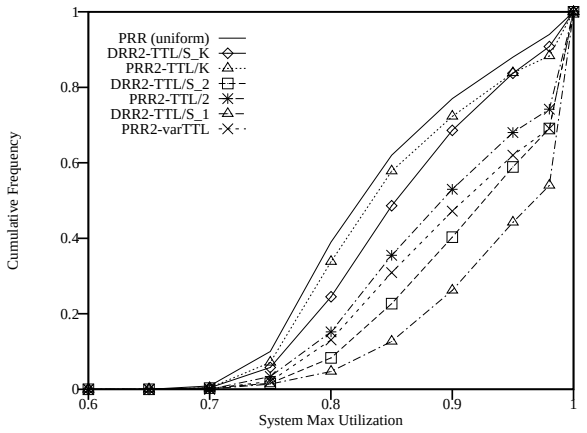

Figure 7: Performance of RR2-based algorithms (Heterogeneity level of 65%)

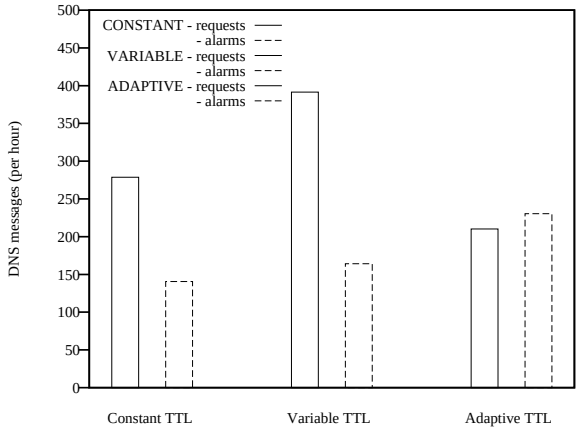

Figure 8: Number of address resolution requests to ADNS

7.3 Sensitivity to system heterogeneity

In this set of experiments we evaluate the sensitivity of the proposed strategies to the degree of system heterogeneity from 20% to 65%. We first focus on deterministic and probabilistic policies (Figure 9 and Figure 10, respectively), and then compare the two classes of policies under RR2 in Figure 11. Now, the y-axis is the probability that the no node of the Web server system is overloaded, while the x-axis denotes the heterogeneity level.

Figures 9-11 show that most adaptive TTL algorithms are relatively stable that is, their performance does not vary widely when the heterogeneity level increases to 50%. After this level, a more sensible performance degradation can be observed for all policies. However, for any heterogeneity level, a large gap exists among the schemes that use a different TTL value for each connected domain and the other policies. Moreover, while achieving the best performance, the TTL/S_K and TTL/K algorithms display the best stability, too. TTL/2 algorithms are still acceptable when combined with RR2-based scheduling schemes, while they tend to degrade more for higher heterogeneity level when they are combined with the RR-based scheduling schemes. The varTTL algorithm, which uses a variable TTL depending upon the number of overloaded nodes, is very unstable as shown in Figure 11, while the DRR-TTL/S_1 strategy (as shown in Figure 9) performs much worse than any other *adaptive TTL* policies and more similar to a *constant TTL* strategy (comparing Figure 9 with Figure 3). Hereafter, we do not consider these two types of strategies, because of their instability and poor performance, respectively. Figure 11 shows that when we assign a different TTL to each connected domain, DRR2-TTL/S_K performs the best, while with only two classes of domains, PRR2-TTL/2 performs better than DRR2-TTL/S_2 for higher heterogeneity levels. From all the shown results, the following are observed.

- *Adaptive TTL* schemes, especially DRR2-TTL/S_K, and PRR2-TTL/K, are very effective in avoiding overloading the nodes even when the system is highly heterogeneous and domain hit rates are unevenly distributed as the pure Zipf's function.
- *Constant TTL* strategies cannot handle the non-uniformity of client distribution as well as node heterogeneity. Various enhancements, such as those provided by restricted-RR and two-alarm-RR schemes, are not really effective.
- Differentiating requests coming from more popular and normal domains improves the performance, regardless of whether TTL is dynamically chosen or fixed. Indeed, RR2-based strategies are always slightly better than their RR-based counterparts.
- Deterministic strategies typically perform better than probabilistic schemes. However the difference is not large and tends to diminish for high heterogeneity levels.

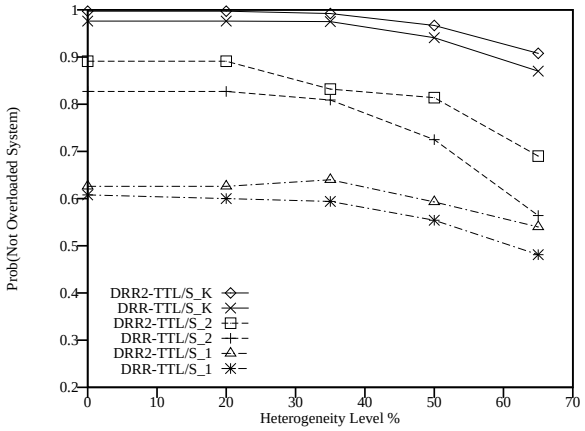

Figure 9: Sensitivity to system heterogeneity of *deterministic* algorithms

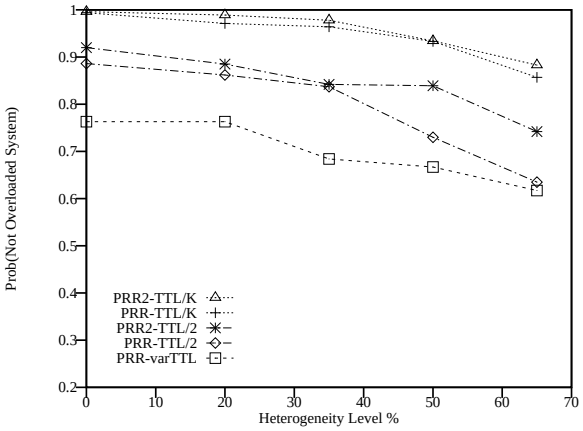

Figure 10: Sensitivity to system heterogeneity of *probabilistic* algorithms

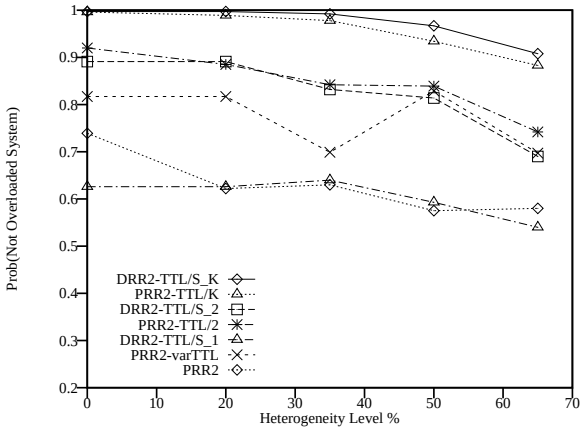

Figure 11: Sensitivity to system heterogeneity of RR2-based algorithms

7.4 Sensitivity to TTL values and workload parameters

We now consider sensitivity to the average TTL values. In Figure 12, both the PRR2-TTL/K and PRR2 are shown with a 20% heterogeneity level for different mean TTL values from 300 to 500 seconds. (DRR2-TTL/S_K schemes which show similar behavior as the PRR2-TTL/K schemes are not shown for readability of the figure.) The adaptive TTL schemes outperform the constant TTL scheme (PRR2) with a wide margin regardless of the TLL values. In Figure 13, the DRR2-TTL/S_K and PRR2 are shown with a 50% heterogeneity level for various mean TTL values. (The PRR2-TTL/K schemes are not shown for readability of the figure.) The superiority of the adaptive TTL schemes is again observed.

Figure 14 shows the sensitivity of the overload probability of the various TTL policies to the mean TTL value, when the heterogeneity level is at 20%. The various dynamic

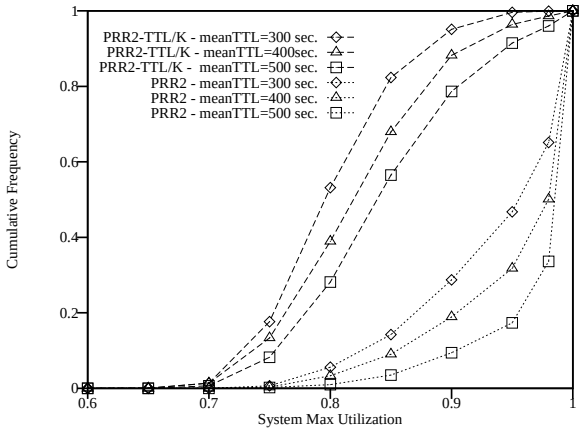

Figure 12: Sensitivity to TTL of *probabilistic* algorithms

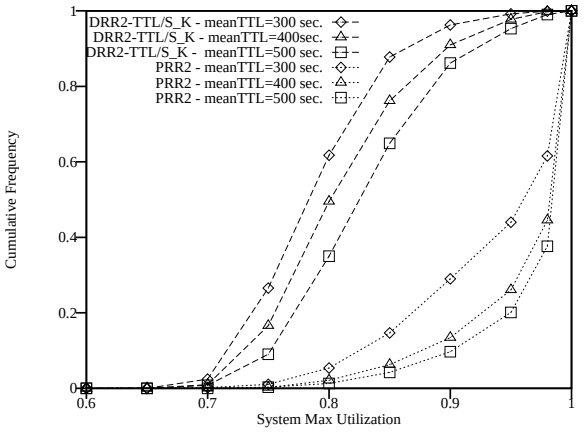

Figure 13: Sensitivity to TTL of *deterministic* algorithms

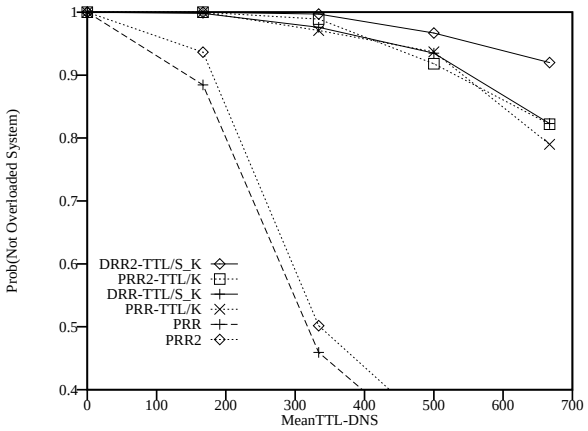

Figure 14: Sensitivity to TTL of Dynamic TTL schemes

TTL schemes perform far superior to the constant TTL schemes, PRR and PRR2, and show much less sensitivity to the TTL values. Among the dynamic TTL schemes, DRR2-TTL/S_K provides the best performance. Also all the RR2 policies perform better than the corresponding RR policies.

We next study the sensitivity to the client request distributions. In addition to the pure Zipf distribution (with the skew parameter $x = 0$), a geometric distribution (with $p = 0.3$), a Zipf distribution with $x = 0.5$, and a uniform distribution (corresponding to $x = 1$) are considered. Figure 15 shows the performance of PRR2-TTL/K under these different client distributions with a heterogeneity level of 35%. The performance improves as the skew in the client distribution decreases. The geometric distribution has performance close to the pure Zipf distribution. We note that the mean TTL value is kept the same at 400 seconds for all cases. Because the client distributions have different skew, the minimum TTL values

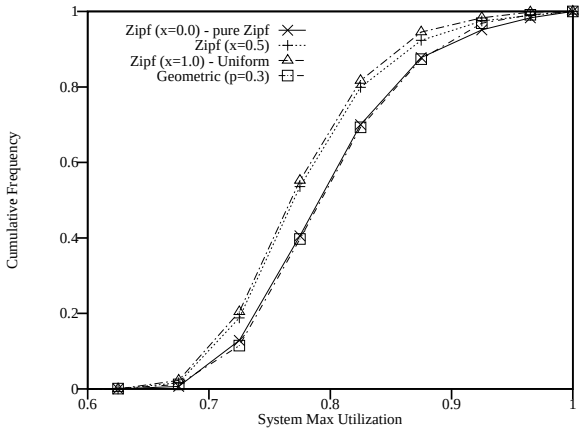

Figure 15: Sensitivity to client distribution (same mean TTL=400 seconds)

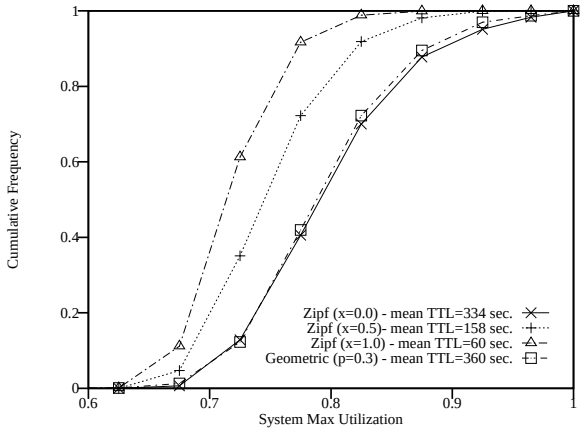

Figure 16: Sensitivity to client distribution (same $\overline{TTL} = 60$ seconds)

under PRR2-TTL/K will be different for the different client distributions (See Equation 3). If we hold the minimum TTL value ($\overline{TTL}$) at 60 seconds for all cases as in Figure 16, the performance gap will be much larger as the skew in the client distribution increases. The spread of the TTL values among the client domains (hence also the mean TTL value) also increases with the skew as indicated in Figure 16.

Next, the sensitivity to the hit service time is examined. We consider the case where some of the hit requests such as CGI dynamic request has particularly long service time. We introduce a new type of Web page requests which consist of one hit of the CGI-type, and five other ordinary hits as considered before, where a CGI-type hit is assumed to have an average service time that is about 10 times the ordinary hits. Figure 17 shows the performance of PRR2-TTL/K under different percentages of this new type of Web page requests containing a CGI-type hit. The dynamic TTL scheme handles workload with long hit service time very well. As the percentage of these long Web page requests increases, the performance actually improves. This is due to the fact that for a given amount of total load to the system, if the per request load increases, the number of subsequent requests arriving during the TTL period will decrease so that the ADNS control improves.

7.5 Robustness of adaptive TTL schemes

The previous results point out a clear preference for *adaptive TTL* schemes. We now examine their robustness by considering two specific aspects. One is the impact of name servers and gateways not following the TTL value recommended by the ADNS. The other is how sensitive the performance is to the accuracy of the estimated domain hit rate. The latter is less of an

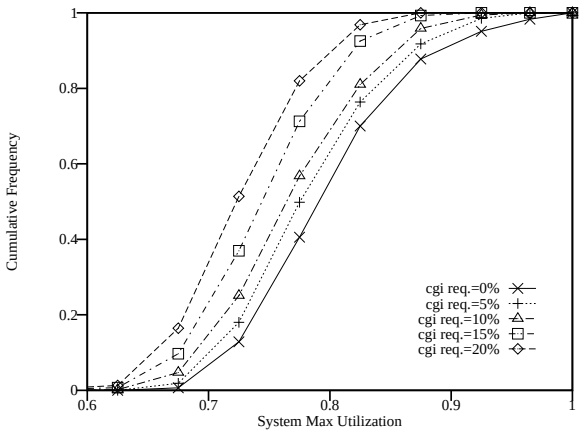

Figure 17: Sensitivity to long requests of PRR2-TTL/K algorithms

issue if the load from each domain remains relatively stable or changes slowly. However, in a more dynamic environment where hit rates from the domains may change continuously, it can be difficult to obtain an accurate estimate.

7.5.1 Effects of non-cooperative name servers

Each name server caches the address mapping for a TTL period. In order to avoid network saturation due to address resolution traffic, very small TTL values are typically ignored by name servers. Since there is not a common TTL lower threshold which is adopted by all name servers, in our study we consider the worst case scenarios, where all name servers and gateways are considered non-cooperative if the proposed TTL is lower than a given minimum, and perform sensitivity analysis against this threshold.

Figure 18 shows the sensitivity of the adaptive TTL policies to the minimum accepted TTL value by the name servers, when the heterogeneity level is at 35%. The performance of DRR2-TTL/S_K and PPR2-TTL/K gradually deteriorates as the minimum TTL value allowed by the name servers increases. However, PRR2-TTL/2 is almost insensitive to the minimum accepted TTL. The advantage from DRR2-TTL/S_K or PPR2-TTL/K diminishes as the minimum TTL value accepted by the name servers increases because the TTL/S_K schemes may sometimes need to select quite a low TTL value when a client request coming from a hot domain is assigned to a node with limited capacity. On the other hand, a probabilistic TTL/2 strategy is almost not affected by the problem of non-cooperative name servers because it uses a rough partitioning (that is, two classes) of the domains and is able to always assign TTL values higher than 180 seconds in all experiments.

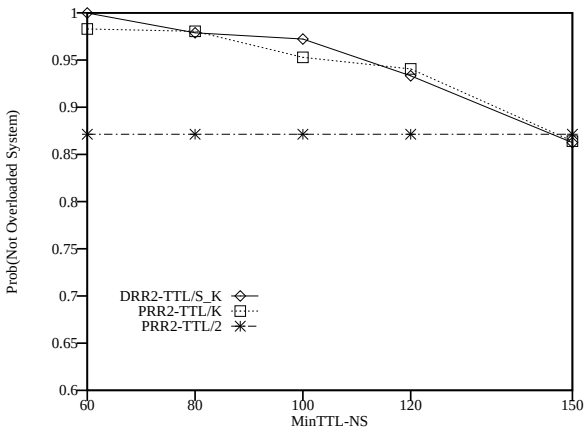

Figure 18: Sensitivity to minimum accepted TTL by name servers

7.5.2 Effects of the estimation error

Next we will examine how the maximum error in estimating the hit rate of each domain may affect the system performance. Figures 19 and 20 compare various adaptive TTL schemes as a function of the estimation error for heterogeneity levels of 20% and 50%, respectively. In the experiment, we introduce a perturbation to the hit rate of each domain, while the ADNS estimates of the domain hit rates remain the same as before. Hence the percentage error in the load estimate is the same as the amount of perturbation on the actual load. For the case of a χ% error, the hit rate of the busiest domain is increased by χ% and the hit rates of the other domains are proportionally decreased to maintain the same total load on the system. This effectively increases the skew of the hit rate distribution, hence represents a worst case.

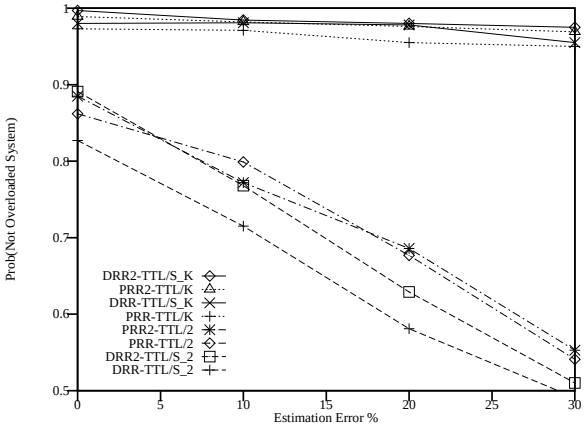

Figure 19: Sensitivity to estimation error (Heterogeneity level of 20%)

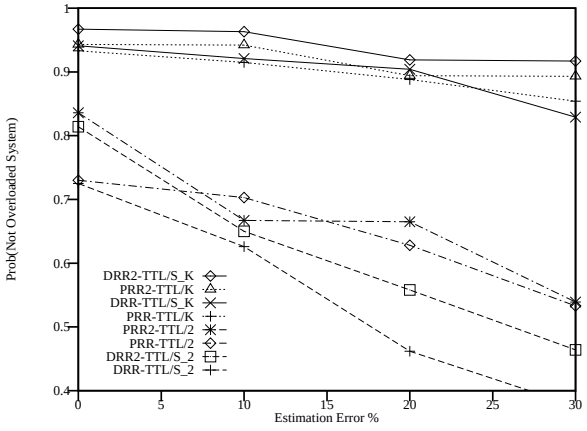

Figure 20: Sensitivity to estimation error (Heterogeneity level of 50%)

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

Although the TTL/S_2 and TTL/2 algorithms are very sensitive to the estimation error (even when this is limited to 10%), it is important to note that the shown results refer to a positive perturbation on the client domain with the heaviest hit rate. This is actually a worst (unrealistic) case because we indirectly increase the skews of the client request distributions to more than a pure Zipf's distribution. Other results not reported, which consider negative perturbation on the heaviest domain hit rate, show analogous performance for the TTL/K policies and much better performance for the TTL/S_2 and TTL/2 strategies. This was expected because a reduction of the hit rate of the busiest domain makes client requests more evenly distributed than a pure Zipf's distribution.

8 Summary of the performance study

Table 2 outlines the specific methods that each ADNS algorithm uses to address heterogeneous nodes and uneven distribution of client requests.

In summary,

- To balance the load across multiple Web nodes, ADNS has two control knobs: the scheduling policy (that is, the node selection) and the TTL value for the period of validity of the selection.
  Just exploring the scheduling component alone (*constant TTL* algorithms) is inadequate to address both node heterogeneity and uneven distributions of clients among domains.

- *Variable TTL* policies that dynamically reduce the TTL when the number of overloaded nodes increases so that more control can be given to the ADNS perform better than constant TTL algorithms. However, this approach is not sufficient to cope with high heterogeneity levels.

Category	*ADNS Algorithm*	*Non-uniform Client Distribution*	*Heterogeneous Nodes*
Constant TTL	DRR	—	—
	DRR2	two-tier domain partition	—
	PRR	—	probabilistic routing
	PRR2	two-tier domain partition	probabilistic routing
	DAL	accumulated hidden load weight	normalized bin
	MRL	residual hidden load weight	normalized (residual) bin
	PRR-Alarm2	—	two alarms, probabilistic routing
	PRR2-Alarm2	two-tier domain partition	two alarms, probabilistic routing
	Restr.-RR2	two-tier domain partition	limited probabilistic routing
Variable TTL	PRR-varTTL	—	probabilistic routing
	PRR2-varTTL	two-tier domain partition	probabilistic routing
Adaptive TTL (*Deterministic*)	DRR-TTL/S_1	—	TTL$\propto$(node capacities)
	DRR-TTL/S_*i*	TTL$\propto$(*i*-classes hit rate)	TTL$\propto$(node capacities)
	DRR2-TTL/S_1	two-tier domain partition	TTL$\propto$(node capacities)
	DRR2-TTL/S_*i*	two-tier domain partition TTL$\propto$(*i*-classes hit rate)	TTL$\propto$(node capacities)
Adaptive TTL (*Probabilistic*)	PRR-TTL/1	(*same policy as PRR*)	
	PRR-TTL/*i*	TTL$\propto$(*i*-classes hit rate)	probabilistic routing
	PRR2-TTL/1	(*same policy as PRR2*)	
	PRR2-TTL/*i*	two-tier domain partition TTL$\propto$(*i*-classes hit rate)	probabilistic routing

Table 2: Summary of ADNS scheduling algorithms.

- *Adaptive TTL* schemes can be easily integrated with even simple scheduling policies such as RR or RR2. This approach shows good performance for various node heterogeneity levels and system parameters, even in the presence of non-cooperative Internet name servers.

- When there is full control on the choice of the TTL values that is, all (or most) name servers are cooperative, DRR2-TTL/S_K is the strategy of choice.

- When there is limited control on the chosen TTL values, both DRR2-TTL/S_K and PRR2-TTL/K show reasonable resilient to the effect of non-cooperative name servers.

- Both DRR2-TTL/S_K and PRR2-TTL/K perform well, even if the domain hit rate cannot be accurately estimated because of high variability of the load sources.

- The heterogeneity level of the Web server system affects the achievable level of performance. Specifically, our results indicate that if the degree of heterogeneity is within 50%, the probability of node overloading guaranteed by best adaptive TTL policies is always less than 0.05-0.10. Therefore, to achieve satisfactory performance, it would be desirable not to exceed this heterogeneity level in the design of a distributed Web

server system.

Moreover, the adaptive TTL algorithms give the best results even for homogeneous Web server systems. In these instances, they are almost always able to avoid overloading nodes even if the ADNS control on the client requests remains below 3-4% of the total load reaching the Web site.

9 Conclusions

Although distributed Web server systems may greatly improve performance and enhance fault-tolerance of popular Web sites, their success depends on load sharing algorithms that are able to automatically assign client requests to the most appropriate node. Many interesting scheduling algorithms for parallel and distributed systems have previously been proposed. However, none of them can be directly adopted for dynamically sharing the load in a distributed Web server system when request dispatching is carried out by the Authoritative DNS of the Web site. The main problems are that the ADNS dispatcher controls only a small fraction of the client requests which actually reach the Web server system. Besides that, these requests are unevenly distributed among the Internet domains. The problems are further complicated when we consider the more likely scenario of a Web server system consisting of heterogeneous nodes.

We first showed that extending known scheduling strategies or those adopted for the homogeneous node case [13] does not lead to satisfactory results. Therefore, we propose a different class of strategies, namely *adaptive TTL* schemes. They assign a different expiration time (TTL value) to each address mapping taking into account the capacity of the chosen node and/or the relative load weight of the domain which has originated the client request.

A key result of this paper is that, in most situations, the simple combination of an alarm signal from overloaded nodes and *adaptive TTL* dramatically reduces load imbalance even when the Web server system is highly heterogeneous and the ADNS scheduler controls a very limited portion of the incoming requests. Moreover, the proposed strategies demonstrate high robustness. Their performance is almost not affected even when the error in estimating the domain load is sizable (say 30%), and it is only slightly affected in the presence of some non-cooperative name servers that is, name servers and gateways not accepting low TTL values that could be sometimes proposed by the ADNS.

Acknowledgements

This work was entirely carried out while Michele Colajanni was a visiting researcher at the IBM T.J. Watson Research Center, Yorktown Heights, NY. This paper benefited from preliminary discussions held with Daniel Dias, IBM T.J. Watson Research Center, and from contributions in experiments given by Valeria Cardellini, University of Roma Tor Vergata, Italy. The authors wish to thank the anonymous referees for their helpful comments on earlier versions of this paper.

References

[1] P. Albitz, C. Liu, *DNS and BIND*, O'Reilly and Associates, Cambridge, MA, 1998.

[2] V.A. Álmeida, I.M.M. Vasconcelos, J.N.C. Arabe, "The effect of the heterogeneity on the performance of multiprogrammed parallel systems", *Proc. Workshop on Heterogeneous Processing*, Beverly Hills, CA, 1992.

[3] Alteon Web Systems, http://www.alteonwebsystems.com/

[4] D. Andresen, T. Yang, O.H. Ibarra, "Towards a scalable didtributed WWW server on networked workstations", *Journal of Parallel and Distributed Computing*, vol. 42, pp. 91-100, 1997.

[5] 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.

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

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

[8] V. Cardellini, M. Colajanni, P.S. Yu, "Dynamic load balancing on Web-server systems", *IEEE Internet Computing*, vol. 3, no. 3, pp.28-39, May-June 1999.

[9] V. Cardellini, M. Colajanni, P.S. Yu, "DNS dispatching algorithms with state estimators for scalable Web-server clusters", *World Wide Web Journal*, Baltzer Science Publ., vol. 2, no. 2, pp. 101-113, July 1999.

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

[11] Cisco's DistributedDirector, http://www.cisco.com/

[12] Cisco's LocalDirector, http://www.cisco.com/

[13] M. Colajanni, P.S. Yu, D.M. Dias, "Analysis of task assignment policies in scalable distributed Web-server systems", *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 6, pp. 585-600, June 1998.

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

[15] 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.

[16] P. Barford, A. Bestavros, A. Bradley, M.E. Crovella, "Changes in Web client access patterns: Characteristics and caching implications", *World Wide Web*, Baltzer Science, vol. 2, no. 1-2, Mar. 1999.

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

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

[19] F5 Networks, http://www.f5labs.com/

[20] R. Fielding, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, June 1999.

[21] Foundry Networks, http://www.foundrynetworks.com/

[22] 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.* (WWW7), Melbourne, Australia, Apr. 1998.

[23] HydraWeb Technologies, http://www.hydraweb.com/

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

[25] 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.

[26] A.K. Iyengar, J. Challenger, D. Dias, P. Dantzig, "High-performance Web site design techniques", *IEEE Internet Computing*, vol. 8, no. 2, pp. 17-26, Mar.-Apr. 2000.

[27] 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", *Jornal of Parallel and Distributed Computing*, vol. 28, pp. 1-18, 1995.

[28] D. Mosedale, W. Foss, R. McCool, "Lesson learned administering Netscape's Internet site", *IEEE Internet Computing*, vol. 1, no. 2, pp. 28-35, Mar.-Apr. 1997.

[29] R. Mukherjee, "A scalable and highly available clustered Web server", in *High Performance Cluster Computing: Architectures and Systems, Volume 1*, Rajkumar Buyya (ed.), Prentice Hall, 1999.

[30] J. Pitkow, "In search of reliable usage data on the WWW", *Proc. of 6th Int. World Wide Web Conference* (WWW6), Santa Clara, CA, Apr. 1997.

[31] Radware Networks, http://www.radware.com/

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

[33] Resonate Global Dispatcher, http://www.resonate.com/

[34] R.J. Schemers, "lbmnamed: A load balancing name server in Perl", *Proc. 9th Systems Administration Conference*, Monterey, CA, Sep. 1995.

[35] T. Schroeder, S. Goddard, B. Ramamurthy, "Scalable Web server clustering technologies", *IEEE Network*, pp. 38-45, May-June 2000.

[36] A. Singhai, S.-B. Lim, S.R. Radia, "The SunSCALR framework for Internet servers", *Proc. of 28-th Int. Symp. on Fault-Tolerant Computing* (FTCS), Munich, Germany, June 1998.

[37] H. Schwetman, *CSIM18: The Simulation Engine*, Mesquite Software Inc., 1999.

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

Figure Captions

- Figure 1. Software component diagram.
- Figure 2. Performance of *constant TTL* algorithms (Heterogeneity level of 20%).
- Figure 3. Sensitivity to system heterogeneity of *constant TTL* algorithms.
- Figure 4. Performance of *deterministic* algorithms (Heterogeneity level of 20%).
- Figure 5. Performance of *probabilistic* algorithms (Heterogeneity level of 20%).
- Figure 6. Performance of RR2-based algorithms (Heterogeneity level of 35%).
- Figure 7. Performance of RR2-based algorithms (Heterogeneity level of 65%).
- Figure 8. Number of address resolution requests to ADNS.
- Figure 9. Sensitivity to system heterogeneity of *deterministic* algorithms.
- Figure 10. Sensitivity to system heterogeneity of *probabilistic* algorithms.
- Figure 11. Sensitivity to system heterogeneity of RR2-based algorithms.
- Figure 12. Sensitivity to TTL of *probabilistic* algorithms.
- Figure 13. Sensitivity to TTL of *deterministic* algorithms.
- Figure 14. Sensitivity to TTL of Dynamic TTL schemes.
- Figure 15. Sensitivity to client distribution (same mean TTL=400 seconds).
- Figure 16. Sensitivity to client distribution (same $\overline{TTL} = 60$ seconds).
- Figure 17. Sensitivity to long requests of PRR2-TTL/K algorithms.
- Figure 18. Sensitivity to minimum accepted TTL by name servers.
- Figure 19. Sensitivity to estimation error (Heterogeneity level of 20%).
- Figure 20. Sensitivity to estimation error (Heterogeneity level of 50%).

Table Titles

- Table 1. Parameters of the system (all time values are in seconds).
- Table 2. Summary of ADNS scheduling algorithms.

Footnote

- In this paper we use the definition of *global scheduling* given in [10], and *dispatching* as its synonymous.

Biographies

Michele Colajanni. Michele Colajanni is currently a Full Professor in the Department of Computer Engineering at the University of Modena, Italy. He received the Laurea degree in computer science from the University of Pisa in 1987, and the Ph.D. degree in computer engineering from the University of Roma Tor Vergata in 1991. From 1992 to 1998, he was at the University of Roma Tor Vergata, Computer Engineering Department as a researcher. He has held computer science research appointments with the (Italian) National Research Council, visiting scientist appointments with the IBM T.J. Watson Research Center, Yorktown Heights, New York. His research interests include high performance computing, parallel and distributed systems, Web systems and infrastructures, load balancing, performance analysis and simulation. In these fields he has served as a member of organizing or program committees of national and international conferences, and has published more than 60 papers in international journals, book chapters and conference proceedings. Michele Colajanni is a member of the IEEE Computer Society and the ACM.

Philip S. Yu. Philip S. Yu received the B.S. degree in E.E. from National Taiwan University, Taipei, Taiwan, Republic of China, in 1972, the M.S. and Ph.D. degrees in E.E. from Stanford University, in 1976 and 1978, respectively, and the M.B.A. degree from New York University in 1982.

Since 1978 he has been with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. Currently, he is manager of the Software Tools and Techniques group. One of the project focuses is to develop algorithms and tools for Internet and E-commerce applications and Web site management, such as the Web usage mining tool in the IBM Domino Go product. Other project activities include intelligent and scalable proxy and Web servers and on-line interactive data mining via Web browsers. Dr. Yus current research interests include data mining, Internet and E-commerce applications, database systems, multimedia systems, and parallel and distributed processing. Dr. Yu has published more than 260 papers in refereed journals and conferences, and over 140 research reports and 90 invention disclosures. He holds or has applied for 85 US patents.

Dr. Yu is a pioneer on the data sharing architecture (for clustering multiple processors) incorporated in the successful IBM System/390 Parallel Sysplex and parallel join algorithms. He had made significant contribution in the area of analytic modeling on database systems, especially in a cluster environment and in applying analysis techniques to analyze various real system problems in the database area, achieve better understanding on the critical performance factors and the tradeoffs of different design

alternatives, and lead to better system or algorithm designs. Large database systems, especially coupled system with a large number of nodes, are extremely time-consuming to simulate. Conventional queueing models are not directly applicable to address the database concurrency control, buffer coherency control issues or buffer hit probabilities. There is a gap between theoretical models and real (database) systems. By developing accurate approximate analytic models, various design alternatives can be quickly evaluated and timely input can thus be provided to the development team to make strategic decisions. The work had made significant impacts to the design of the IBM System/390 Parallel Sysplex. Dr. Yu has received two Outstanding Innovation awards from IBM on his contributions to the Parallel Sysplex design and development. In the multimedia area, Dr. Yu has also made fundamental contributions on the scheduling of continuous media streams. These include the group sweeping scheme for disk arm scheduling, various load balancing schemes and novel approaches to support VCR functions. He has received numerous patents in this area.

Dr. Yu is a Fellow of the ACM and a Fellow of the IEEE. He was an editor of IEEE Transactions on Knowledge and Data Engineering and also a guest co-editor of special issue on mining of databases. In addition to serving as program committee members on various conferences, he was the program co-chair of the 11th Intl. Conference on Data Engineering, and the program chair of the 2nd Intl. Workshop on Research Issues on Data Engineering: Transaction and Query Processing. He served as the general chair of the 14th Intl. Conference on Data Engineering. He has received several IBM and external honors including Best Paper Award, 2 IBM Outstanding Innovation Awards, Outstanding Technical Achievement Award, 2 Research Division Awards, and 21 Invention Achievement Awards.

Complete Contact Information

Michele Colajanni	
Mailing address	Dipartimento di Ingegneria Informatica Università di Modena Modena, Italy 41100
E-mail	colajanni@unimo.it
Phone	+39-059-2056137
Fax	+39-059-2056129

Philip S. Yu	
Mailing address	IBM T. J. Watson Research 30 Saw Mill River Road Hawthorne, NY 10514
E-mail	psyu@us.ibm.com
Phone	(914) 784 7141
Fax	(914) 784 7141