Redirection Algorithms for Load Sharing in Distributed Web-server Systems[*]

Valeria Cardellini
Università di Roma Tor Vergata
Roma, Italy 00133
cardellini@uniroma2.it

Michele Colajanni
Università di Modena
Modena, Italy 41100
colajanni@unimo.it

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

Abstract

Replication of information among multiple Web servers is necessary to support high request rates to popular Web sites. A clustered Web-server organization is preferable to multiple independent mirrored-servers because it maintains a single interface to the users and has the potential to be more scalable, fault-tolerant and better load balanced. In this paper we propose a Web cluster architecture in which the Domain Name System *server (DNS), that dispatches the user requests among the servers through the URL-name to IP-address mapping mechanism, is integrated with a redirection request mechanism based on the HTTP protocol. This should alleviate the side effect of caching the IP-address mapping at intermediate name servers. We compare many alternative mechanisms, including synchronous vs asynchronous activation, and centralized vs distributed decision on redirection. Moreover, we analyze reassignment of entire domains or individual client requests, different types of status information, and different server selection policies for redirecting requests. Our results show that the combination of centralized and distributed dispatching policies allows the Web-server cluster to handle the high load skews in the WWW environment.*

1. Introduction

A common approach adopted by popular Web sites to handle millions of accesses per day is to preserve one virtual URL interface and use a distributed server architecture which is hidden from the user. This system provides scalability and transparency, but requires some internal mechanism that dynamically assigns client requests to the Web server that can offer the best service [5, 8]. The assignment decision can be taken at the *IP level* through some address packet rewriting mechanism [8, 11], or at the *Domain Name System* (DNS) level through the mapping of the URL-name to the IP-address of one of the servers in the cluster [1, 4, 5]. Both choices have some drawbacks. The IP-dispatcher based systems have full control on the incoming requests, but they can be applied only to locally clustered Web servers. (The exception is the Network Dispatcher approach which can support multiple Network Dispatchers [11].) Moreover, the task of rewriting all packets can cause the IP-dispatcher to become a bottleneck when the system is subject to heavy request load. The DNS-dispatcher based clusters do not present risks of bottleneck, and can easily scale from locally to geographically distributed Web-server systems. The main problem of scheduling through the DNS is due to the IP-address caching mechanism that lets the DNS control only a very small fraction of the user requests. The limited control and the high non-uniformity of the load from different client domains require sophisticated DNS scheduling policies to avoid Web server overload [5, 6].

In this paper, we will focus on an alternative architecture that integrates the DNS dispatching mechanism with a redirection technique carried out by the Web servers through the redirection mechanism provided by HTTP [9]. Such redirection is transparent to the users that at most perceive a small increase in the response time. Unlike the IP-dispatcher based solutions, the HTTP redirection does not require the modification of the IP-address of the packets reaching or leaving the Web-server cluster. We propose and evaluate a large set of alternative redirection schemes. We demonstrate that the DNS-dispatcher combined with suitable redirection mechanisms provides excellent load control that minimizes server overload.

The paper is organized as follows. Section 2 discusses

[*] ©1999 IEEE. Published in the *Proceedings of IEEE 19th Int. Conf. on Distributed Computing Systems* (ICDCS'99), Austin, TX, May 1999. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 732-562-3966.

the system model and the design space for the redirection schemes. Section 3 and 4 presents various policies in which redirection decisions are periodically taken by the DNS or asynchronously activated by the Web servers, respectively. Section 5 presents the experimental results and Section 6 analyzes related work. Section 7 concludes the paper with some final remarks.

2. System model and alternative redirection schemes

The users access the WWW services through some client application. Typically, the clients have a (set of) local name server(s) and are connected to the network through local gateways. We will refer to the network sub-domain behind these local gateways as *domain*. The Web-server cluster uses one URL-name to provide a single interface for viewers. The system consists of homogeneous distributed servers that manage the same set of Web documents, and a (primary) DNS that translates the URL-name into the IP-address of one of the servers in the cluster. Through this mechanism, the DNS can dispatch the requests among the servers based on some optimization criterion. Besides the non-uniform distribution of client requests among the domains, the main DNS problem is that IP-address caching at local and intermediate name servers limits the control of the DNS to a small fraction of the requests reaching the Web cluster. That is to say, during the *time to live* (TTL) period for caching the IP-address mapping at intermediate name servers, bursts of requests can arrive from a domain to the same server, thereby causing high load skews [7], especially if the domain has a large number of clients.

To solve these problems, we integrate the DNS-dispatcher with some redirection techniques carried out by the Web servers through the HTTP protocol. Various alternatives are possible. We analyze those that are fully compatible with existing Web standards and protocols. In particular, the DNS and Web servers of the cluster are the only entities here that collect and exchange load information.

A summary of the major factors of the design space of the redirection schemes analyzed in this paper is given in Table 1. The first group of factors is on activating the redirection process, including the *activation trigger* mechanism (synchronous vs asynchronous) in which a redirection decision process is activated, and the *activation decision* process (centralized vs distributed). The second group is on *status information* used to implement the redirection scheme. This can either be the Web server load information and/or the domain load information. The third group is on carrying out the redirection policy. This includes the redirection server selection for receiving the redirected requests, and the entities that are redirected which can be an entire domain and/or some individual clients within a domain. The details of the various alternatives will be explained in later sections.

We classify the different redirection approaches based on the activation trigger mechanism and activation decision process. Here we assume that the synchronous (or periodic) activation is always combined with a centralized decision by the DNS, while an asynchronous activation comes always together with a distributed decision by any of the Web servers. Once a redirection decision has been made, the redirection process is always carried out by the Web servers. Hence, we can group the alternative approaches into two main classes: *centralized synchronous redirection* (in brief, synchronous redirection), where the decision about redirection is taken periodically at the DNS; *distributed asynchronous redirection* (in brief, asynchronous redirection), where the redirection decision process can be activated by any Web server that is critically loaded.

We describe various algorithms with synchronous and asynchronous activation of the redirection scheme. The classification takes into account the other factors presented in Table 1. Finally, we comment on the status information. In this paper, the server load information is the utilization over a short interval, while the domain load information is measured as a *domain hit rate*, i.e., the number of hits per second reaching the Web-server cluster from a domain.

3. Synchronous redirection policies

The synchronous redirection algorithms have the following common features. The decision is centralized at the DNS. Every t seconds each Web server sends some status information (server and/or domain load) to the DNS. The DNS gathers information from all the servers and builds the so called *Assignment Table*, where it specifies for each connected domain the assigned Web server. The DNS serves the address resolution requests by using this table.

The redirected entities can be entire domains (DR), individual clients (CR) or both (CDR). As the HTTP redirection mechanism works on an individual basis, *domain redirection* means that all clients of the same domain are subject to the same redirection decision.[1] In DR and CDR policies, the DNS broadcasts the Assignment Table to all the Web servers. For each client request, the Web server checks the current Assignment Table to verify if it has to serve or to redirect the requests coming from that domain.

[1]Redirection is done by indicating, on the header of a response from a server, the IP-address of the new server and the code 301 (i.e., Server Moved Permanently) [9]. The IP-address cache of the client that receives this response is automatically modified, hence all subsequent requests of the session from this client will go to the newly assigned server. On the other hand, the other clients of the same domain are not affected by this redirection reply, because the IP-address cache of the domain gateway has not been modified. Hence, when an entire domain is redirected, the HTTP redirection mechanism would need to redirect every client request from that domain.

Parameter	*Alternatives*		
Activation trigger (*when*)	Synchronous (*periodic*)		Asynchronous (*on server request*)
Activation decision (*where*)	Centralized (*DNS-dispatcher*)		Distributed (*Web servers*)
Status information	Server load (*server utilization*)	Alarm	Domain load (*domain hit rate*)
Server selection (*how*)	Assignment Table	Assignment Table and Server Percentage List	Available Server List
Redirected entities (*what*)	Domains (DR)	Clients (CR)	Clients and Domains (CDR)

Table 1. Alternative design choices for redirection schemes.

3.1. Domain redirection

We now consider the case where the redirection entity is the entire domain. Here we assume a system in which every Web server periodically transmits some status information to the centralized (DNS) dispatcher. The first interesting issue to examine is the implication of different status information on the algorithms that build the Assignment Table. We analyzed various schemes that used just request load from each domain, just server load, or both information. Space limits do not allow us to describe the details of the carried out analysis. The main conclusion is that any scheme that builds the Assignment Table using the domain or server load information alone performs much more poorly than schemes that use both sources of information. Hence, in this paper we will focus on domain and server load information based algorithms.

The best domain redirection (**DR**) scheme works as follows. As first step, the DNS estimates the domain hit rate for each connected domain, and on this basis it orders the domains from the most to the least popular. Then, through some server load information, the DNS orders the servers from the least to the most loaded. In the third step, the DNS builds the Assignment Table through the domain hit rate information. It determines the potential load assigned to each server through a bin, that contains the sum of the hit rates of the domains assigned to that server by the Assignment Table under construction. The server bin is updated based on the domain load after each assignment. The assignment policy aims at equalizing the bin levels through a greedy approach. In the first phase, the most popular domain is assigned to the least loaded server and so on until each server gets one domain. The other domains are assigned by selecting each time the server with the lowest bin level.

3.2. Client redirection

When the DNS decides (re)assignments of entire domains to servers, it is almost impossible to get a perfect balancing, because the domain hit rates present high skews especially for the hottest domains. A finer grain redirection could be achieved by working on individual clients instead of entire domains. The class of *client redirection* algorithms builds the Assignment Table following the same method of the DR scheme. This table is now used only in the first level assignment carried out by the DNS when it receives an address resolution requests.

The redirection or second level assignment carried out by the Web servers is instead based on the so called *Server Percentage List* to indicate the percentage of client requests that need to be redirected. This is built by the DNS in the following way. The DNS first estimates the average bin level across all servers. In the Server Percentage List, the servers with bin level below the average or within an acceptable range will have a server percentage set to 0%. For the other servers, the DNS evaluates the percentage of additional load exceeding the average as its server percentage to be reassigned. For example, let us suppose that the server WS_2 has a bin level which is 30% higher than the average. WS_2 is assigned with a server percentage equal to 30% in the Server Percentage List.

Once obtained the Server Percentage List, the DNS broadcasts it to each server. This list is used for implementing the following probabilistic redirection mechanism based on individual clients. The Web servers that have a server percentage equal to 0% do not reassign any request. A Web server with a percentage higher than 0% at each page request generates a random number p uniformly distributed between 0 and 1. If it comes higher than its server percentage (considering as example WS_2, if $p > 0.3$), the server will return the required information. Otherwise, it redirects the requests coming from that client to another server.

We consider three possibilities for the choice of the server that has to receive these redirected requests. In **CR_RR** policy, client requests are reassigned in a cyclic way to all servers with percentage equal to 0% in the Server Percentage List. In **CR_PRR** policy, client redirection is done in a probabilistic round-robin (PRR) way, where the probability is based on the available server capacity using the latest server load information. This information can be easily broadcasted by the DNS together with the Server Percentage List. Just for comparison purpose, we consider the simple scheme that reassigns the client requests to the server which is indicated as the the least loaded (LL) by the last DNS broadcast. This is referred to as **CR_LL**.

3.3. Domain and client redirection

If the redirection of entire domains is a too coarse intervention on the load distribution, and the redirection of individual clients could not work for the opposite reason, the third alternative is to combine both methods. A mechanism that redirects domains and individual clients requires both the Assignment Table and Server Percentage List. Now the Assignment Table is used not only by the DNS for the first level assignment but also by the Web servers for the second level (re)assignment. For this reason, the DNS has to periodically broadcast both of them to the Web servers.

At the arrival of a client request, each Web server checks the current Assignment Table to verify if it has to serve the requests coming from that domain. If not, the server redirects the requests according to the Assignment Table. Otherwise it checks the Server Percentage List. If its percentage is equal to 0%, the server returns the required information. Otherwise, it implements one of the redirection mechanisms presented in the previous section, i.e., CDR_RR, CDR_PRR or CDR_LL.

3.4. Alarm messages

Any of the previously presented synchronous algorithms can be combined with a feedback alarm mechanism. When a server finds that it is becoming overloaded, it sends an alarm message to the DNS. The DNS excludes this server from further assignments in the Assignment Table until it receives another message from the same server that signals the return to a normal load status. The algorithms that use this alarm message mechanism are denoted by the same abbreviation name plus the keyword ‘alarm’.

4. Asynchronous redirection schemes

The feedback alarm mechanism outlined in Section 3.4 can be used to activate the redirection process itself. This would lead to a new class of distributed reassignment schemes that are asynchronously activated on a Web server request. No Assignment Table needs to be generated. The Web cluster remains a typical DNS-dispatcher based system where the DNS carries out the first level assignment through a RR or RR2 scheme [5] where two independent round-robin schedules are maintained in RR2: one for the heavily loaded domains, and the other for the remaining domains. This DNS assignment process is integrated with a second level (re)assignment mechanism triggered by any overloaded server.

The DNS now maintains the so called *Available Server List* which is the list of servers that are not overloaded at that moment. This list is transmitted in reply to a server alarm message that the server sends to the DNS when its utilization has exceeded a given load threshold. Each overloaded server may redirect its client requests to any server on the Available Server List through the same HTTP-based protocol used by the synchronous algorithms. The selection of a server from the Available Server List can be done in different ways. Any simple algorithm such as round-robin or random can be used. The redirection mechanism can be considered purely distributed because the DNS has now taken the simple role of information collector/communicator.

The client redirection activated by heavily loaded servers can overcome the so called TTL constraint. In fact, with IP-address caching at intermediate name servers, DNS-dispatcher loses direct control on the subsequent client requests to the assigned server for the TTL period following the URL-name to IP-address assignment. So it takes longer for the overloaded server to recover because the DNS policy can only stop the new DNS assignments to the overloaded server. There is no means to remove the already made assignments until the TTL expires. With redirection, the over-utilized server can get rid of a fraction of the previously assigned requests before TTL expires.

5. Experimental Results

5.1. Simulation model and parameters

We assume that clients are partitioned among the domains based on a Zipf’s distribution, i.e. a distribution where the probability of selecting the i-th domain is proportional to $1/i^{(1-x)}$. Indeed, if one ranks the popularity of domains by the frequency of their accesses to the Web server, the distribution on the number of clients in each domain is a function with a short head (corresponding to big providers, organizations and companies), and a very long tail [2]. In most of the experiments the clients are partitioned among the domains based on a pure Zipf’s distribution; in Section 5.4 we show a sensitivity analysis to Zipf’s parameter.

We consider major components that impact the performance of the Web-server cluster. This includes the intermediate name servers that affect operations and performance of the DNS dispatching algorithms through their IP-address caching mechanisms. We consider all the details concerning a client *session* that is the entire period of access to the Web site from a single user. Each session consists of two phases: the *IP-address request* phase during which the client asks the DNS for a translation of the Web cluster URL into the IP-address of one of the Web servers in the cluster; the *Web page request* phase in which various pages are requested directly from the Web server selected by the DNS. The

IP-address request is initially submitted to the local name server of the client domain, because it typically caches the URL-name to IP-address mapping for the TTL period. If the cache of the local name server has a valid mapping for this URL-name, the page request is sent directly to the Web server without going through the DNS request. Otherwise, the IP-address request is submitted to subsequent intermediate name servers, and only if the mapping is not cached in any of these name servers, the request reaches the DNS of the Web-server cluster. The DNS returns the IP-address of one of the servers and the TTL value. Each name server along the path from the DNS to client's domain caches this mapping for the TTL period.

The number of page requests per session (with a mean of 12 pages/session) and the time between two page requests from the same client (with a mean of 25 seconds) are assumed to be exponentially distributed [2]. Since an HTML page is typically composed of a collection of objects, each of them requires an access to the server (*hit*). The number of hits per page are obtained from a uniform distribution in the discrete interval (5—15) [7]. The hit service time and the inter-arrival time of hit requests to the servers are assumed to be exponentially distributed. In the experiments, the cluster average utilization is always kept to 2/3 of the whole capacity in all experiments. This value is obtained as a ratio between system load, i.e., the total number of hits per second arriving to the Web cluster, and the cluster capacity, i.e., the sum of the server capacities denoted in hits per second. There are in average 2500 clients distributed among 50 connected domains. All DNS policies use a TTL value set to 300 seconds. The Web cluster is assumed to consist of 7 servers with a total capacity to handle 1500 hits/sec.

The main goal of this study is to investigate the impact of the redirection algorithms on avoiding that some Web server becomes overloaded. Therefore, we define the *cluster maximum utilization* at a given instant as the highest server utilization at that instant among all Web servers. Specifically, the major performance criterion is the *cumulative frequency* of the cluster maximum utilization, i.e., the probability (or fraction of time) that the cluster maximum utilization is below a certain value. By focusing on the highest utilization among all Web servers, we can deduce whether the Web cluster is overloaded or not.

In some figures showing sensitivity to other system parameters, we use the probability that no server of the Web cluster is overloaded as the performance metric. This can be the 95 percentile of the cluster maximum utilization (or not exceeding 95% utilization, respectively). The simulators, based on the Independent Replication Method, were implemented using the CSIM package. Each value is the result of five or more simulation runs with different seeds, where each run is for six hours of the Web-server system activity. For all simulation results, confidence intervals were estimated and the 95% confidence interval was estimated to be within 5% of the mean.

5.2. Synchronous redirection performance

In synchronous redirection schemes, the DNS has to collect some status information to build the Assignment Table. Every t seconds each Web server communicates server and domain load information to the DNS that replies through a broadcast of the Assignment Table. This interval t is referred to as *Assignment Table update* interval and is set to 60 seconds in the first set of experiments.

We found that any scheme that builds the Assignment Table using the domain or server load information alone shows very poor performance. For most of them, the probability that no server is overloaded is around or below 0.2. This result is close to the **Random** algorithm that provides a random assignment of the domains to the servers in the table. On the other hand, building the Assignment Table through a combination of domain and server information improves the performance substantially. Figure 1 shows that the domain redirection scheme (DR) has a probability of 0.9 of not causing any Web server to exceed 95% utilization.

The next question to be investigated is to determine the best redirection granularity. A granularity of redirection finer than DR is achieved by client redirection schemes (CR), and client and domain redirection schemes (CDR), that reassign individual clients and both entire domains and individual clients, respectively. Figure 1 summarizes the performance of CR and CDR schemes, where the Web servers can use round robin (RR) or least loaded server (LL) algorithms to select the server to which redirecting the client requests. The LL algorithm does not perform well as it fails to spread the load among multiple servers. In not shown experiments we found that there is no appreciable difference between RR and probabilistic RR (PRR) schemes. As

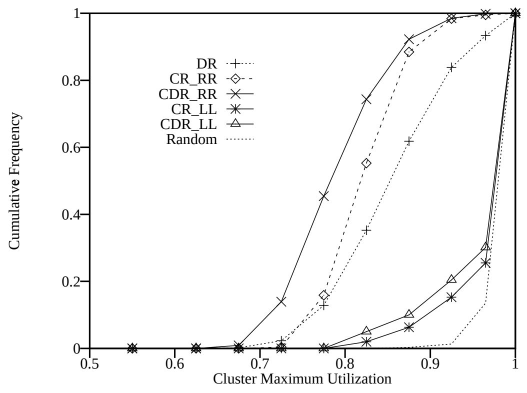

Figure 1. Comparison of best performing synchronous schemes.

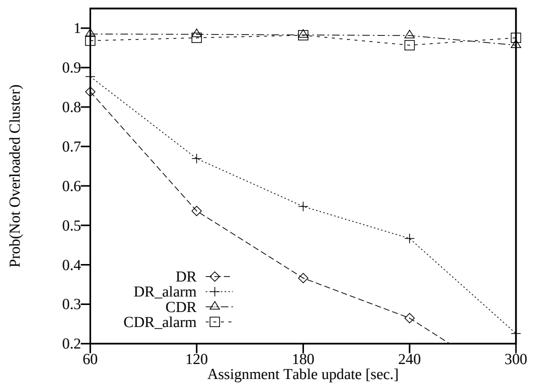

Figure 2. Sensitivity of DR and CDR to the frequency of Assignment Table updating.

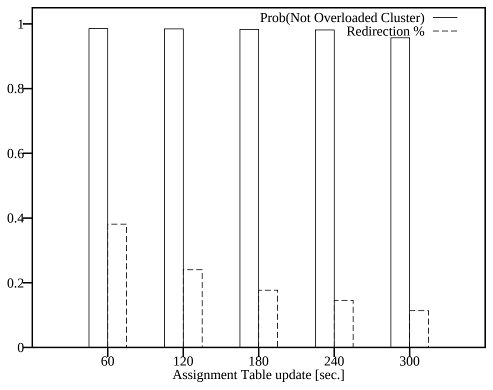

Figure 3. Sensitivity of CDR to the frequency of the Assignment Table updating.

PRR requires additional information with no performance improvement, we use CR_RR and CDR_RR as the basic client redirection, and client and domain redirection algorithm, respectively. Henceforth, we will focus on the CDR scheme because Figure 1 shows that client redirection combined with domain redirection is better than client redirection alone and improves substantially the performance of the domain redirection (DR) scheme.

We now consider how it is possible to minimize the overheads of the synchronous schemes, i.e. to reduce DNS-servers communications and client request reassignments. Indeed, if the period of activation is short, the synchronous redirection algorithms can cause high computation and communication overheads due to gathering status information, building the Assignment Table and broadcasting it to the servers. Hence the main goal is to reduce the frequency of updating the Assignment Table. Actually, we found that for the CR scheme the best updating interval is a value close to the TTL chosen by the DNS (in this paper typically set to 300 seconds). As this interval is sufficiently high to limit communication overheads and the percentage of requests redirected by CR is only about 4-5%, the optimization analysis is focused on DR and CDR schemes.

In Figure 2 we compare the sensitivity to the Assignment Table update interval using the probability that no server of the cluster is overloaded (exceeding 95% utilization) as the performance metric. We consider the best DR and CDR schemes and their combination with the feedback alarm at DNS (as discussed in Section 3.4). The alarm threshold is set to 0.85. Even though the exclusion of overloaded servers from the Assignment Table (DR_alarm scheme) improves the results of the basic DR algorithm, the performance becomes very poor as the update interval increases. On the other hand, the CDR schemes with or without alarm are quite insensitive to the Assignment Table update interval. This stability is very important because an increase of the update interval allows a reduction of not only the computation/communication overhead but also the number of reassigned requests. So we can limit the percentage of users that perceive an increase in the response time without affecting the performance for the CDR scheme.

For the CDR scheme, Figure 3 shows that increasing from a 60 seconds to a 300 seconds update interval causes only a very limited performance degradation (the probability that no server is overloaded drops slightly from 0.99 to 0.96), but a substantial reduction of reassigned requests (from 0.38 to 0.12).

5.3. Asynchronous redirection performance

In asynchronous redirection schemes the DNS has to collect alarm messages from heavily loaded servers. The utilization threshold that triggers the alarm message is set to 0.75. In the shown results, each server evaluates if its utilization has exceeded the alarm threshold every 16 seconds, referred to as the *check-load interval*.

An asynchronous client redirection algorithm (ACR) can be described by specifying the first level assignment scheme carried out by the DNS and the second level (re)assignment algorithm executed by the Web servers. However, as there is not much differences among the performance of different server selection policies (RR or PRR) for redirecting requests, in this section we distinguish the ACR algorithms on the basis of the *DNS assignment scheme*. In particular, we consider round-robin (ACR_RR), round-robin combined with alarm from heavily loaded servers (ACR_RR_alarm), and two-tier round-robin with alarm (ACR_RR2_alarm). These alarm messages exclude the overloaded servers even from the DNS assignments. (For additional details about these DNS policies, the reader can refer to [5].)

In Figure 4 we compare the performance of these ACR schemes with that of RR and RR2 with alarm where the

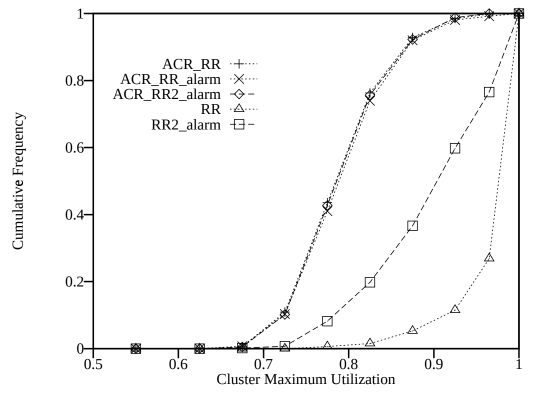

Figure 4. Performance of asynchronous schemes with various DNS algorithms.

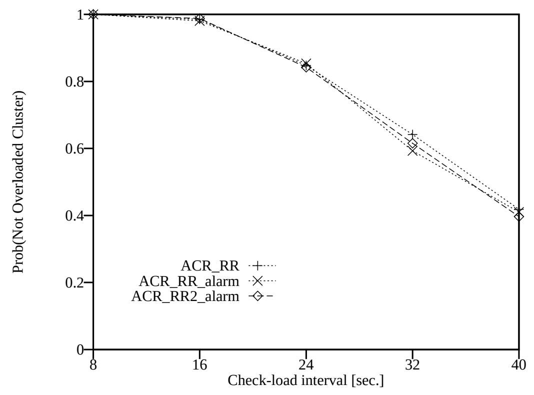

Figure 5. Sensitivity of asynchronous schemes to the check-load interval.

DNS first level assignment is not integrated with a second level server reassignment. The improvement in favor of any ACR algorithm is considerable. The client redirection can overcome the drawbacks caused by the IP-address caching mechanisms. Even stateless schemes, such as RR that was shown to perform very poorly under skewed workload on client distributions, when combined with a client redirection mechanism (e.g., ACR_RR) have performance better than stateful schemes (e.g., RR2 with alarm [5]).

However, Figure 5 shows that all asynchronous redirection schemes are very sensitive to the length of the *check-load interval*. Although the results deteriorate as the check-load interval increases, it is reasonable to use short periods such as 16 seconds because the server load evaluation does not necessary imply an activation of the redirection mechanism. Furthermore, it does not incur communication overhead. This is in contrast to shortening the Assignment Table update interval in the synchronous redirection case.

5.4. Sensitivity analysis

We now compare the best centralized synchronous CDR algorithm with the best distributed asynchronous ACR algorithm for various system scenarios. The performance comparison is carried out on the two most critical parameters: sensitivity to system utilizations (Figure 6) and sensitivity to client distributions among the domains (Figure 7). Changing other system parameters such as average number of requests, user think time or user client session time did not show noticeable differences among the redirection approaches, hence it is not presented here. Figure 6 compares the performance of a system where the utilization varies from 0.5 to 0.75. The CDR algorithm shows better results than the asynchronous ACR approach.

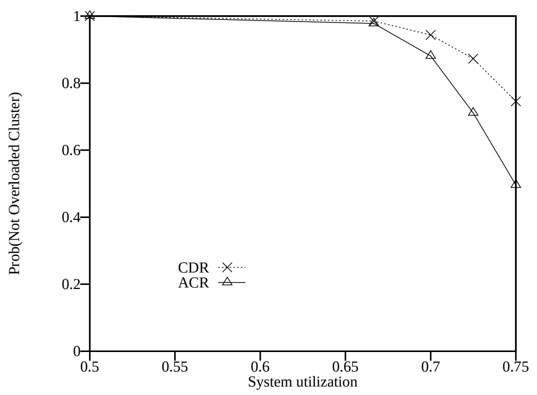

Figure 6. Sensitivity to system utilization.

Analogous conclusion can be done when we vary the distribution of the clients among their domains. Figure 7 shows the probability that no server has a utilization higher than 0.9 as a function of the Zipf parameter (the x-axis goes from a pure Zipf distribution to the uniform distribution). Actually, we have to change the performance metric (0.9 instead of the usual 0.95 adopted in this paper) to show some

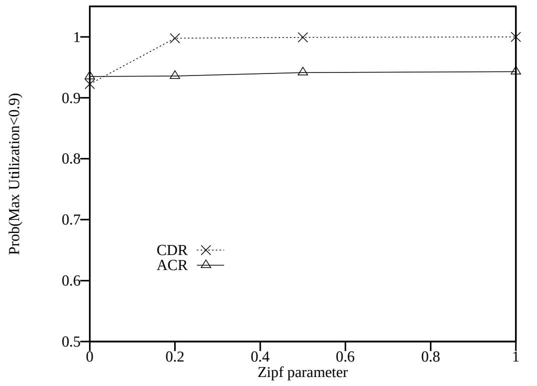

Figure 7. Sensitivity to client distribution.

difference between CDR and ACR, because both perform well. The robustness with respect to the client distribution is important because in the real WWW environment the client scenarios tend to change frequently. Another confirmation that both CDR and ACR perform well is given by other experimental measures showing that the percentages of redirected requests are below 0.2 for both approaches.

6. Related Work

Various request redirection techniques have already been proposed for distributed Web-server clusters. Both the Cisco DistributedDirector (CiscoDD) [4] and the Distributed Server Groups (DSG) dispatcher [10] are based on a totally centralized approach. All the requests reaches the dispatcher that uses the HTTP redirection mechanism to redirect the requests to the most appropriate Web server. The dispatcher of DSG selects the least loaded server [10], while that of CiscoDD uses some network metric to estimate the server with the closest proximity to the client that has originates a request [4]. Both DSG and CiscoDD use redirection as the only means to dispatch requests. There is not another dispatching level as in our proposals.

Closer to our algorithms that use a two-level dispatching (DNS plus redirection) are the SWEB [1] and *Distributed Packet Rewriting* (DPR) [3] systems. In SWEB, client requests are initially assigned from the DNS in the round-robin manner. Then, each server may reassign a received request to any other server of the cluster through the HTTP protocol. The decision to serve or to redirect a request is based on the criteria to minimize the response time, which is evaluated through the server processing capability and Internet bandwidth/delay. DPR uses simpler decision criteria such as a hash function applied to the client packet address or the least loaded server. Unlike all other approaches, DPR does not use the HTTP protocol but a packet rewriting mechanism to reroute client requests.

7. Conclusions

Replication of information among multiple Web servers is necessary to support high request rates to popular Web sites. In this paper, we have studied Web cluster architectures in which the DNS dispatcher function is integrated with some redirection mechanism carried out by the servers. We have compared various alternative mechanisms with synchronous or asynchronous activation, and centralized or distributed decision on redirection.

Our performance results demonstrate that the DNS scheduling policies integrated with some redirection mechanisms are effective, even in the presence of highly skewed load. The experiments indicate that the most useful status information to decide about reassignment is a combination of domain load and server load. Moreover, the redirection of individual client connections is necessary to balancing the load better than domain reassignment alone. However, there are significant differences between asynchronous and synchronous schemes. Individual client redirection is sufficient to achieve acceptable performance for asynchronous schemes, whereas it works less well for synchronous algorithms unless it is combined with domain redirection.

The centralized synchronous algorithm gives the best results for a wide set of system parameters. However, the performance difference with distributed asynchronous policies is not appreciable unless the Web-server cluster is subject to very heavy load. Moreover, the intra-cluster communication overhead of synchronous algorithms is typically higher than that introduced by asynchronous policies.

Acknowledgments

The first two authors acknowledge the support of the Ministry of University and Scientific Research in the framework of the project “Design Methodologies and Tools of High Performance Systems for Distributed Applications”.

References

[1] D. Andresen, T. Yang, V. Holmedahl, O.H. Ibarra, “SWEB: Toward a scalable World Wide Web server on multicomputers”, *Proc. of IPPS’96*, Honolulu, April 1996.

[2] M.F. Arlitt, C.L. Williamson, “Web server workload characterization: The search for invariants”, *IEEE/ACM Trans. on Networking*, vol. 5, no. 5, Oct. 1997.

[3] A. Bestavros, M. E. Crovella, J. Liu, D. Martin, “Distributed Packet Rewriting and its application to scalable Web server architectures”, *Proc. of 6th IEEE Int’l. Conf. on Network Protocols*, Austin, TX, Oct. 1998.

[4] Cisco’s DistributedDirector, http://www.cisco.com/warp/public/751/distdir/

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

[6] M. Colajanni, P.S. Yu, V. Cardellini, “Dynamic load balancing in geographically distributed heterogeneous Web-servers”, *Proc. of ICDCS’98*, Amsterdam, May 1998.

[7] C. Cunha, A. Bestavros, M. Crovella, “Characteristics of WWW client-based traces”, Tech. Rep. BUCS-95-010, Boston Univ., April 1995.

[8] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, “A scalable and highly available Web server”, *Proc. of 41st IEEE Computer Society Int’l. Conf.*, Feb. 1996.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2068, Jan. 1997.

[10] M. Garland, S. Grassia, R. Monroe, S. Puri, “Implementing Distributed Server Groups for the World Wide Web”, Tech. Rep. CMU-CS-95-114, Carnegie Mellon Univ., Jan. 1995.

[11] G.D.H. Hunt, G.S. Goldszmidt, R.P. King, R. Mukherjee, “Network Dispatcher: A connection router for scalable Internet services”, *Proc. of WWW7*, Brisbane, Apr. 1998.