Scalable Web Clusters with Static and Dynamic Contents

Emiliano Casalicchio
University of Rome Tor Vergata
Roma, Italy, 00133
ecasalicchio@ing.uniroma2.it

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

Abstract

Cluster systems are leading architectures for building popular Web sites that have to guarantee scalable services when the number of accesses grows exponentially. The most common Web cluster systems consist of replicated back-end and Web servers, and a Web switch that routes client requests among the nodes. In this paper, we propose a new scheduling policy, namely Multi-Class Round Robn (MC-RR), for Web switches operating at layer 7 of the OSI protocol stack. Its goal is to improve load sharing in recent Web clusters that provide multiple services such as static and dynamic information.

We demonstrate through a wide set of simulation experiments that other dispatching policies aiming to improve locality in server caches give best results for traditional Web publishing sites providing static information and some simple database searches. On the other hand, when we consider more recent Web sites providing dynamic and secure services, MC-RR is much more effective than state-of-the-art Web switch policies. The proposed algorithm has the additional benefit of guaranteeing more stable results than those of most dynamic policies whose performance depend on optimal tuning of several parameters, very hard to achieve in highly variable Web systems.

1 Introduction

Most popular Web sites are suffering from severe congestion, since they are getting thousands of requests every second in coincidence or not with special events. Moreover, the heterogeneity and complexity of services and applications provided by Web server systems is continuously increasing. Traditional Web publishing sites with most static contents have being integrated with recent Web commerce and transactional sites combining dynamic and secure services. The most obvious way to cope with growing service demand and application complexity is adding hardware resources because replacing an existing machine with a faster model provides only temporary relief from server overload. Furthermore, the number of requests per second that a single server machine can handle is limited and cannot scale up with the demand. The need to optimize the performance of Web services is producing a variety of novel architectures.

Geographically distributed Web servers and proxy server systems aim to decrease user latency time through network access redistribution and reduction of amount of data transferred, respectively. In this paper we consider different Web systems, namely *Web clusters*, that use a tightly coupled distributed architecture. Cluster technology is a cost effective solution because it is cheaper than a single faster machine. From the user's point of view, any HTTP request to a Web cluster is presented to a logical (front-end) server that acts as a representative for the Web site. This component called *Web switch* retains transparency of the parallel architecture for the user, guarantees backward compatibility with Internet protocols and standards, and distributes all client requests to the Web and back-end servers. Cluster architectures with Web switch dispatcher(s) have been adopted with different solutions in various academic and commercial Web clusters. A valuable recent survey is in [13].

One of main operational aspects of any distributed system is the availability of a mechanism that shares the load over the server nodes. Numerous *global scheduling*[1] algorithms were proposed for multi-node architectures executing parallel or distributed applications. Most of these policies were analyzed under the hypothesis of Poissonian task arrivals and exponential service times. Unlike traditional distributed systems, a Web cluster is subject to quite different workload. Internet traffic tends to occur in waves with intervals of heavy peaks, moreover the service time of HTTP requests may have very large or infinite variance, especially when the Web cluster provides heterogeneous services. The load characteristics reaching a Web cluster have motivated proposals of global scheduling strategies based on dynamic server state information. However, dynamic Web switch policies require expensive and hard to tuning mechanisms for monitoring and evaluating the load

[1] In this paper we use the definition of global scheduling given in [6].

on each server, gathering the results, combining them, and taking real-time decisions. For these reasons, we propose a *multi-class round robin policy* (MC-RR) policy that takes dynamic decisions by looking more at client requests than at server states. MC-RR partitions the Web cluster services into broad classes on the basis of the expected impact that each request may have on main server components that is, network interface, CPU, disk. Then, it schedules client load by taking into account the service class each request belongs to. Under workload characteristics that resemble those experienced by real Web sites, we demonstrate that this simple client request-aware policy is much more effective than state-of-the-art dynamic algorithms when applied to Web clusters providing heterogeneous services such as static, dynamic and secure information. By using MC-RR, the 90-percentile of page request time can be half of that of commonly used dynamic Web switch policies, such as Weighted-Round Robin [10] and LARD [12]. Moreover, MC-RR guarantees stable results for a wide range of Web sites because it does not need a hard tuning of parameters for each type of Web site as most dynamic policies require.

The remaining part of this paper is organized as follows. In Section 2, we outline the typical architecture of a Web cluster and give some examples of real systems and Web switches. In Section 2.3, we discuss some global scheduling algorithms for the Web switch and describe the MC-RR policy. In Section 4, we present a detailed simulation model for the Web cluster and the parameters of the workload models. In Section 5, we discuss the experimental results for various classes of Web sites. In Section 6, we give our final remarks.

2 Web clusters

2.1 Architecture

A Web cluster refers to a Web site that uses two or more server machines housed together in a single location to handle user requests. Although a large cluster may consist of dozens of Web servers and back-end servers, it uses one hostname to provide a single interface for users. To have a mechanism that controls the totality of the requests reaching the site and to mask the service distribution among multiple servers, Web clusters provide a single virtual IP address that corresponds to the address of front-end server(s). Independently of the system mechanism that existing Web clusters use to routing the load, we refer to this entity as the *Web switch*. The Domain Name Server(s) for the Web site translates the hostname for the site into the IP address of the Web switch. In such a way, the Web switch acts as a centralized global scheduler that receives the totality of the requests and routes them among the servers of the cluster (see Figure 1).

We consider a Web cluster consisting of homogeneous distributed servers that provide the same set of documents and services. The details about the operations of the Web cluster are described in Section 4. Various academic and commercial products confirm the increasing interest in these distributed Web architectures. In the *IBM TCP router* [9], all HTTP requests reach the Web switch that distributes them by modifying the destination IP address of each incoming packet: the Web switch replaces its IP address with the private address of the selected Web server.

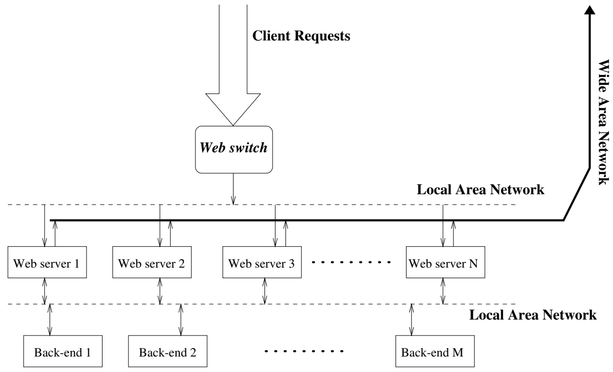


Figure 1. Web cluster architecture.

Magicrouter [1], *Distributed Packet Rewriting* [5] and Cisco *LocalDirector* [7] are other Web cluster architectures relying on a Web switch that receives the totality of client requests. In particular, Magicrouter is a mechanism of fast packet interposing where a user level process acting as a switchboard intercepts network packets and modifies them by changing addresses and checksum fields. Cisco LocalDirector rewrites the IP header information of each incoming packet according with a dynamic table of mapping between each session and the server to which it has been redirected. Unlike the TCP router that modifies only the client-to-server packets and lets the servers modify outgoing IP packets, Magicrouter and LocalDirector Web switches can be defined as *gateways* because they intervene even on server-to-client packets.

An evolution of the TCP router architecture is represented by the IBM *Network Dispatcher* that does not require a modification of the packet addresses because packet forwarding to cluster nodes is done at the MAC address level [10]. A different forwarding approach to configure a Web system with multiple servers uses the if-config-alias option, which is available in most UNIX platforms [8]. This architecture publicizes the same secondary IP address of all Web servers as the IP single virtual address, namely *ONE-IP*, of the Web cluster. This is achieved by letting the servers of the cluster share the same IP address as their secondary address, which is used only for the request distribution service.

2.2 Web switches

A key component of any Web cluster is the Web switch that dispatches client requests among the servers. They can be broadly classified according to the OSI protocol stack layer at which they operate, so we have *layer-4* and *layer-7* Web switches [13].

Layer-4 Web switches work at TCP/IP level. Since packets pertaining to the same TCP connection must be assigned to the same server node, the Web switch has to maintain a binding table to associate each client TCP session with the target server. The switch examines the header of each inbound packet and on the basis of the flag field determines if the packet pertains to a new or an existing connection. Layer-4 Web switch algorithms are *content information blind*, because they choose the target server when the client establishes the TCP/IP connection, before sending out the HTTP request. Global scheduling algorithms executable at the layer-4 Web switch range from static algorithms (say, random, round-robin) to dynamic algorithms that take into account either network client information, (say, client IP address, TCP port), or server state information (say, number of active connections, least loaded server) or even a combination of both information.

Layer-7 Web switches can establish a complete TCP connection with the client and inspect the HTTP request content prior to decide about dispatching. In such a way, they can deploy *content information aware* distribution, by letting the Web switch examine the HTTP request and then route it to the target server. The selection mechanism (usually referred to as *delayed binding*) can be based on the Web service/content requested, as URL content, SSL identifiers, and cookies. In [2] there are many techniques to realize the dispatching granularity at the session level or at the single Web object request level. Scheduling algorithms deployed at layer-7 may use either client information (as session identifiers, file type, file size [11]) or a combination of client and server state information. The potential advantages of layer-7 Web switches include the possibility to use specialized Web server nodes and partition the Web content among clusters of heterogeneous servers [15], and to achieve higher cache hit rates, for example through affinity-based scheduling algorithms such as the LARD policy [12].

2.3 Web switch algorithms

The real alternative for layer-4 and layer-7 Web switches remains the kind of system information they can use to take assignment decisions.

The Web switch cannot use highly sophisticated algorithms because it has to take fast decision for dozens or hundreds of requests per second. To prevent the Web switch to become the primary bottleneck of the Web cluster, static algorithms are the fastest solution because they do not rely on the current state of the system at the time of decision making. Typical examples are *Random* (RAN) and *Round-Robin* (RR) algorithms. RAN distributes the arrivals uniformly through the nodes. RR uses a circular list and a pointer to the last selected server to take scheduling decisions.

On the other hand, static algorithms can potentially make poor assignment decisions. Dynamic algorithms have the potential to outperform static algorithms by using some state information to help dispatching decisions. On the other hand, dynamic algorithms require mechanisms that collect, transmit and analyze state information, thereby incurring in potentially expensive overheads. We consider the following classes of dynamic algorithms:

- *Server state aware policies* route requests on the basis of some server state information, such as load condition, response time, availability or network utilization. In this paper, we consider the weighted round-robin policy (WRR) described in [10] that can operate at layer-4 Web switches.

- *Client state aware policies* route requests on the basis of some client information. Layer-4 Web switches can use only *network client information* such as client IP address and TCP port. On the other hand, layer-7 Web switches can examine the entire HTTP request and can take decisions on the basis of more detailed information about the client, say *client info aware policies*. The main goal of the proposed policies is to augment disk cache hit rates, for example through the LARD policy [12] or other affinity-based scheduling algorithms [14, 16]. Actually, most of the proposed *client info aware policies* use also some information about the server loads, thereby belonging to the *client and server state aware policies*.

In this paper, we focus two widely used dynamic scheduling policies: *Weighted Round Robin* (WRR) and *Locality Aware Request Distribution* (LARD), as representative examples of layer-4 and layer-7 Web switches algorithms, respectively.

WRR comes as a variation of the round robin policy. WRR associates to each server a dynamically evaluated weight that is proportional to the server load state. Every T_{gat} seconds, the Web switch gathers this information from servers and computes periodically the weights. WRR is actually a class of dynamic policies that uses some information about the system state. The first issue that needs to be addressed when we consider a server state aware policy is how to compute the load state information because it is not immediately available at the Web switch. The three main factors that affect the response time are loads on CPU, disk and network resources. Typical load measures are the number of active processes on server, mean disk response time, and hit response time that is, the mean time spent by each request at the server. In particular, the load indexes we consider are the number of active processes at each server (*WRR_num* policy), and the mean service time for the requests (*WRR_time* policy). Additional information on WRR can be found in [10].

The LARD policy is a content based request distribution that aims to improve the cache hit rate in Web cluster nodes. The principle of LARD is to direct all requests for a Web object to the same server node. This increases the likelihood to find the requested object into the disk cache of the server node. We use the LARD version proposed in [12] with the multiple hand-off mechanism defined in [2] that works for the HTTP/1.1 protocol. LARD assigns all requests for a target file to the same node until it reaches a certain utilization threshold. At this point, the request is assigned to a lowly loaded node, if it exist, or to the least loaded node. To this purpose, LARD defines two threshold parameters: T_{low} denoting the upper bound of a lowly loaded condition, and T_{high} denoting the lower bound of a highly loaded condition.

3 Multi-class Round Robin policy

In this section, we propose a *multi-class round robin policy* (MC-RR) that takes into account some information associated to client requests as it can be get by a layer-7 Web switch. MC-RR in its initial form is a client info aware policy, however it can be easily combined with some server state information.

The basic idea comes from the observation that WRR and LARD policies work fine in Web clusters that host traditional Web publishing sites. In fact, most load balancing problems occur when the Web site hosts heterogeneous services that make an intensive use of different Web server's components. Let us classify Web services into four main categories:

Web publishing sites providing static information (e.g., static HTML pages with some embedded objects) and dynamic services (e.g., result or product display requests) that do not intensively use Web server resources. The content of dynamic requests is not know at the instant of a request, however it is generated from database queries whose arguments are known before hand.

Web transaction sites providing dynamic content generated for personalization of information or from (possibly complex) database queries built from user data provided by an HTML form. These sites can require disk and/or CPU bound services.

Web multimedia sites providing streaming audio and video services. In this paper, we do not consider this type of applications that often are implemented through specialized servers and network connections.

Although realistic, this classification is done for the purposes of our paper only and does not want to be a precise taxonomy for all Web services. The idea behind the MC-RR policy is that, although the Web switch can not estimate precisely the service time of a request, it can distinguish from the URL the class of the request and its impact on Web server components. Any Web content provider can easily tune the CAP policy at its best. Starting from the above classification, we distinguish the Web requests into four classes: *static* and *lightly dynamic* Web publishing services (N); *disk bound* services (DB), *CPU bound* (CB) and *disk and CPU bound* (DCB) for example in Web transaction sites.

In the basic version of MC-RR, the Web switch manages a circular list of assignments for each class of Web services. The goal is to share multiple load classes among all servers so that no single component of a node is overloaded. When a request arrives, the Web switch parses the URL and selects the appropriate server. We describe the MC-RR behavior through the following example. We suppose that the server A has still in queue three requests of type N, one of type DB and two of type CB; the server B has received two requests of type N, and one each for type DB, CB, DCB. The sequence of successive requests at the Web cluster is shown in Figure 2. By using MC-RR, the Web switch assigns the first two requests to the server B, the successive three requests to the server A, and the last request again to the server B. After the assignment, server A and B have a similar number of requests for each class of service.

MC-RR does not require a hard tuning of parameters which is typical of most dynamic policies because the service classes are decided in advance, and the scheduling choice is determined statically once the URL content has been classified.

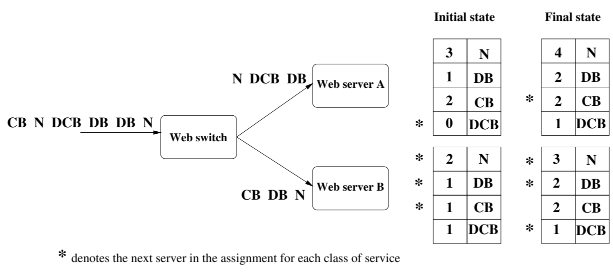


Figure 2. The Multi-Class Round Robin policy.

4 Test-bed model

The Web cluster consists of multiple Web servers and back-end servers, and a dedicated machine that acts as a layer-4 or layer-7 Web switch. The primary DNS translates the hostname of this site into the IP address of the Web switch. The addresses of Web and back-end servers are private and invisible to the extern. Web servers, back-end servers and Web switch are connected through a local fast Ethernet with 100 Mbps bandwidth, as in Figure 1. As the focus is on Web cluster performance we did not model the details of the external network. To prevent the bridge(s) to the external network to become a potential bottleneck for the Web cluster throughput, we assume that the system is connected to the Internet through one or more large bandwidth links that do not use the same Web switch connection to Internet.

Each server in the cluster is modeled as a separate component with its CPU, central memory, hard disk and network interface. All above components are CSIM processes having their own queuing systems that allow for requests to wait if the server, disk or network are busy. We use real parameters to setup the system. For example, the disk is parameterized with the values of a real fast disk (IBM Deskstar34GXP) having transfer rate equal to 20 MBps, controller delay to 0.05 msec., seek time to 9 msec., and RPM to 7200. The main memory transfer rate is set to 100MBps. Network interface is a 100Mbps Ethernet card. In some experiments, a portion of the main memory space of each server is used for Web caching.

Back-end servers are replicated database servers that reply to dynamic requests. The Web server software is modeled as an Apache-like server, where an HTTP daemon waits for requests of client connections. As required by the HTTP/1.1 protocol, each HTTP process serves all files specified in a Web request. The client-server interactions are modeled at the details of TCP connections including packets and ACK signals. Each client is a CSIM process that, after activation, enters the system and generates the first request of connection to the Web switch of the Web cluster. The entire period of connection to the site, namely *Web session*, consists of one or more page requests to the site. At each request, the Web switch applies some routing algorithm and assigns each connection to a server. The server dedicates a new HTTP process for that connection. Each client request is for a single HTML page that typically contains a number of embedded objects. A request may include some computation or database search, and a secure connection. The client will submit a new request only after it has received the complete answer that is, the HTML page and all (possible) embedded objects. A user think time between two page requests models the time required to analyze the requested object and decide about a new request.

Parameter	*Value*
Number of servers	2-32
Disk transfer rate	20 MBps
Memory transfer rate	100 MBps
HTTP protocol	1.1
Intra-servers bandw.	100 Mbps
Arrival rate	100-1600 clients per second
Requests per session	Inverse Gaussian ($\mu = 3.86$, $\lambda = 9.46$)
User think time	Pareto ($\alpha = 1.4$, $k = 2$)
Embedded objects	Pareto ($\alpha = 1.1 - 1.5(1.33)$, $k = 1$)
Hit size - body	Lognormal ($\mu = 7.640$, $\sigma = 1.705$)
Hit size - tail	Pareto ($\alpha = 1.383$, $k = 2924$)

Table 1. Parameters of the system and workload model.

It is worth to observe that for a fair comparison of the Web switch algorithms our models consider the overhead difference in dispatching requests at layer-4 and layer-7 Web switches. For example, the delays introduced by layer-7 switching are modeled with the values given in [2].

Special attention has been devoted to the workload model that incorporates all most recent results on the characteristics of real Web load. The high variability and self-similar nature of Web access load is modeled through heavy tail distributions such as Pareto, lognormal and Weibull distributions [4]. A summary of the distributions and parameters used in our experiments is in Table 1.

To characterize the different Web services classified as in Section 3 we have modeled also the impact of CPU bound requests and intensive database queries (that is, disk bound requests).

5 Experimental results

As the main measure for analyzing the performance of the Web cluster we use the cumulative frequency of the *page response time* because mean values have a little interest in this highly variable environment. The goal of this paper on Web cluster allows us to not include in page response time all delays related to Internet.

We use a different metric when we want to show the difficulty of optimal tuning parameters of some dynamic policies. In particular, we use the *90-percentile* of page response time that is, the page response time limit that the Web site guarantees with 0.9 probability. We compare performance of different scheduling policies for Web clusters under three main classes of workload:

Web publishing site containing *static* and *lightly dynamic* documents. A static document resides on the disk of

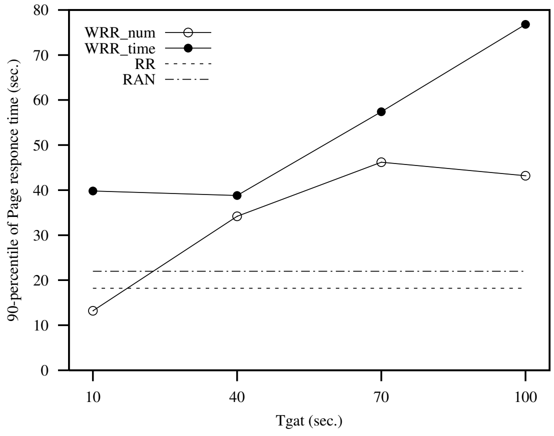
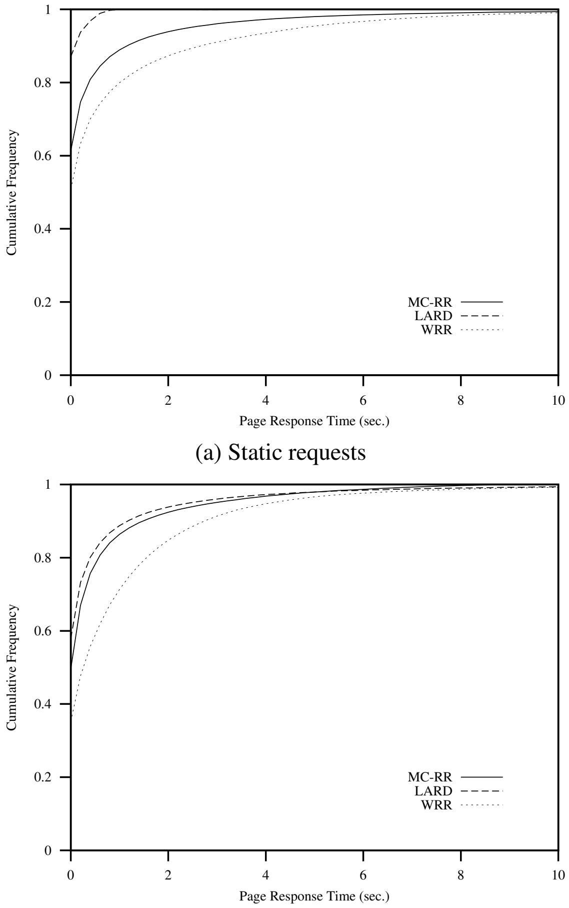


Figure 3. Sensitivity of WRR policies to the T_{gat} parameter and load metric (Web transaction site).



(a) Static requests

(b) Dynamic requests

Figure 4. Web publishing site.

the Web server, it is not modified in a relatively long time interval and is always cacheable. The cache of each node is set to 15% of the total size of the Web site document tree. A lightly dynamic document is cacheable with 0.3 probability.

Web transaction (light) sites contain 60% of static documents and 40% of dynamically created documents. The database query to back-end servers requires intensive disk use and its result is not cacheable.

Web transaction (heavy) sites have 30% of static requests 30% of lightly dynamic requests and various combinations for the remaining 40% of requests, typically with a strong impact on the CPU and/or disk.

5.1 Optimal tuning of dynamic policies

Most dynamic policies depend on system state information that is not immediately available at the Web switch. It is often expensive to gather server state information, and it is difficult to choose the best parameters of dynamic policies in highly variable and heterogeneous Web systems.

The MC-RR policy, in its simplest form, does not require to set any parameter other than the choice of classes of services that can be identified from the URL. The LARD strategy requires to set only the parameters T_{low} and T_{high} for server utilization as described in Section 2.3. In our experiments we set T_{low}=0.3 and T_{high}=0.7, and we did not observed considerable changes for lightly different values, such as 0.2 and 0.75.

The most complex policy to tune is the WRR that is sensible to the adopted load metric (number of active connections, mean page response time at each server) and to the selected T_{gat} value for gathering server state information.

Figure 3 shows the sensitivity of WRR_num and WRR_time with respect to the T_{gat} period for Web transaction site with highly variable workload. The T_{gat} value has a big influence on performance especially for sites with heterogeneous services. If not well tuned, a dynamic policy such as WRR can behave worse than static policies such as RAN and RR. Here and in other not reported experiments the number of active connections (WRR_num) seems to be the best load metric and low T_{gat} values seem to be preferable to higher values. In the remaining part of the paper, we use the number of active processes as a load metric and assume that we are always able to choose the best parameters for the WRR policy. We will refer to it simply as WRR.

5.2 Web publishing

We assume two scenarios where first all requests are for static documents and then for lightly dynamic documents.

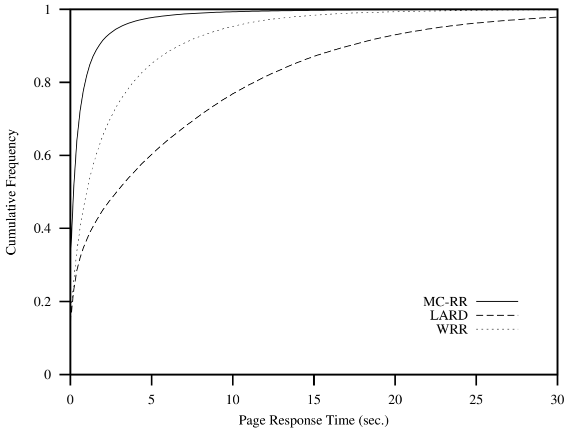


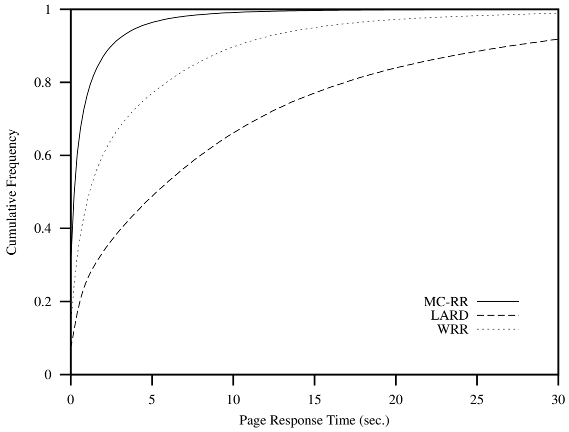
Figure 5. Light Web transaction site

Figure 4(a) shows that the LARD strategy that exploits the reference locality of Web requests performs better than MC-RR and WRR. LARD guarantees with very high probability that the page response time is less than 1 second. The analogous probability is about 0.9 for MC-RR and 0.8 for WRR. In the next experiments we consider only requests for lightly dynamic documents, that have a lower probability of being found in the disk cache. Figure 4(b) shows that MC-RR and LARD have a similar behavior even if LARD still performs slightly better. However, if we compare this figure with Figure 4(a) we can observe that, while MC-RR and WRR maintain similar results, actually LARD is truly penalized by a lower cache hit rate.

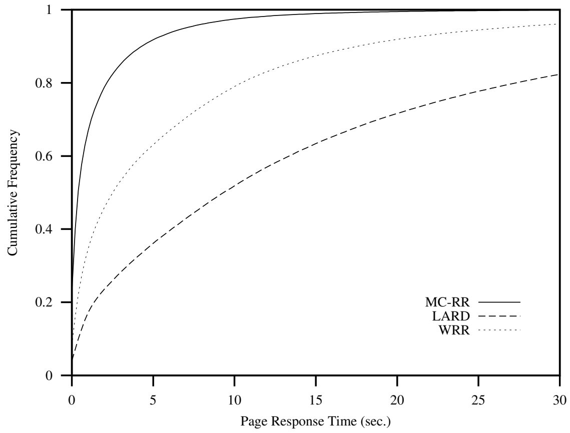
5.3 Web transaction

5.3.1 Light requests

When the document hit rate is low and a relatively high percentage of requests (40%) is disk bound, the MC-RR policy starts to exploit its multiple service characteristics. Figure 5 shows that the page response time of MC-RR is much better than that achieved by WRR and LARD strategies. MC-RR policy improves Web cluster performance of 35% over WRR and 55% over LARD. Indeed, the 90-percentile of page response time using MC-RR is less than 2 seconds, while it is about 7 seconds for WRR and about 20 seconds for LARD. LARD has the worst performance because incoming requests are assigned to a Web server first on the basis of document locality and then on the basis of server load state. Our results show that routing requests to the same server until it reaches a highly loaded state (e.g., T_{high}=0.7) does not work in Web transaction sites.



(a) CPU and disk bound requests



(b) CPU and/or disk bound requests

Figure 6. Heavy Web transaction site.

5.3.2 Heavy requests

This is the most heterogeneous workload. We consider three scenarios where lightly dynamic, CPU bound and DISK bound requests are mixed in different ways.

In the first set of experiments we introduce 20% of CPU bound requests and 20% of disk bound requests. In Figure 6(a) we see that the MC-RR policy provides really good page response times, while WRR is at its limit and LARD does not guarantee a scalable Web cluster. The improvement of the MC-RR strategy is considerable, especially if we consider that the WRR curve refers to the WRR policy with best parameters. For example, MC-RR achieves a page response time of about 4 seconds with 0.95 probability. For WRR, the analogous response time is achieved with a probability of about 0.78, and for LARD with a probability of less than 0.5.

In the last set of experiments, we consider a realistic Web transaction site where the requests can be for CPU bbound services (20%), disk bound (10%) or both (10%) services.

Even if the workload becomes more onerous, Figure 6(b) shows that the MC-RR policy guarantees a scalable Web cluster. Indeed, it achieves a 90-percentile for the page response time of less than 5 seconds. The analogous percentile is equal to 20 seconds for WRR and it is higher than 35 seconds for LARD. For example, WRR and LARD have a page response time of 5 seconds with a probability of 0.66 and 0.37, respectively. This is about two third and one third of the corresponding MC-RR probability.

6 Conclusions

Web cluster architectures are becoming very popular for supporting Web sites with large numbers of accesses and/or heterogeneous services. In this paper, we propose a new scheduling policy, called *client aware policy* (MC-RR), for Web switches operating at layer 7 of the OSI protocol stack to route requests reaching the Web cluster. We demonstrate that dynamic dispatching policies that improve Web caches hit rates give best results for traditional Web publishing sites providing most static information and some lightly dynamic requests. On the other hand, only MC-RR provides scalable performance for recent Web clusters with static, dynamic and secure contents. Moreover, MC-RR has the additional benefit of guaranteeing robust results for very different classes of Web services because it does not require a hard tuning of system parameters as many other dynamic scheduling policies do.

References

[1] E. Anderson, D. Patterson, E. Brewer, "The Magicrouter, an application of fast packet interposing", unpublished Tech. Rep., Computer Science Department, University of Berkeley, May 1996.

[2] M. Aron, P. Druschel, W. Zwaenepoel, "Efficient support for P-HTTP in cluster-based Web servers", *Proc. USENIX 1999*, Monterey, CA, June 1999.

[3] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, "Scalable content-aware request distribution in cluster-based network servers", *Proc. USENIX 2000*, San Diego, CA, June 2000.

[4] P. Barford, A. Bestavros, A. Bradley, M.E. Crovella, "Changes in Web client access patterns: Characteristics and caching implications", *World Wide Web*, Special Issue on Characterization and Performance Evaluation, Jan.-Feb. 1999.

[5] A. Bestavros, M. E. Crovella, J. Liu, D. Martin, "Distributed Packet Rewriting and its application to scalable server architectures", Tech. Rep. BUCS-TR-98-003, Computer Science Department, Boston University, Dec. 1997.

[6] 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, Feb. 1988, pp. 141-154.

[7] Cisco's LocalDirector. Available online at http://cio.cisco.com/warp/public/751/lodir/index.shtml

[8] O.P. Damani, P.E. Chung, Y. Huang, C. Kintala, Y.-M. Wang, "ONE-IP: Techniques for hosting a service on a cluster of machines", *Proc. of 6th Intl. World Wide Web Conf.*, Santa Clara, CA, Apr. 1997.

[9] 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), Feb. 1996, pp. 85-92.

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

[11] M. Harchol-Balter, M. Crovella, C. Murta, "On choosing a task assignment policy for a distributed server system", *J. of Parallel and Distributed Computing*, Vol. 59, pp. 204-228, 1999.

[12] Vivek S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, "Locality-aware request distribution in cluster-based network servers", *In Proc. of 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, Oct. 1998.

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

[14] J. Song, E. Levy-Abegnoli, A. Iyengar, D. Dias, "Design alternatives for scalable Web server accelerators", *Proc. 2000 IEEE Int'l Symp. on Performance Analysis of Systems and Software*, Austin, TX, Apr. 2000.

[15] C.-S. Yang, M.-Y. Luo, "A content placement and management system for distributed Web-server systems", *Proc. of IEEE 20th Int. Conf. on Distributed Computing Systems* (ICDCS'2000), Taipei, Taiwan, Apr. 2000.

[16] X. Zhang, M. Barrientos, J.B. Chen, M. Seltzer, "HACC: An architecture for cluster-based Web servers", *Proc. 3rd USENIX Windows NT Symposium*, Seattle, Washington USA , July 1999.