Content-aware Dispatching Algorithms for Cluster-based Web Servers *

Emiliano Casalicchio [a] Valeria Cardellini [a] Michele Colajanni [b]

[a] *University of Roma Tor Vergata, Roma, Italy 00133*
E-mail: {ecasalicchio, cardellini}@ing.uniroma2.it
[b] *University of Modena, Modena, Italy 41100*
E-mail: colajanni@unimo.it

Cluster-based Web servers are leading architectures for highly accessed Web sites. The most common Web cluster architecture consists of replicated server nodes and a Web switch that routes client requests among the nodes. In this paper, we consider content-aware Web switches that can use application level information to assign client requests. We evaluate the performance of some representative state-of-the-art dispatching algorithms for Web switches operating at layer 7 of the OSI protocol stack. Specifically, we consider dispatching algorithms that use only client information as well as the combination of client and server information for load sharing, reference locality or service partitioning.

We demonstrate through a wide set of simulation experiments that 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, dispatching policies that aim to share the load are the

most effective.
Keywords: Load sharing, Dispatching algorithms, Clusters, World Wide Web, Performance evaluation.

AMS Subject classification: 68M14, 68M20, 68W01.

1. Introduction

The overall growth of traffic on the World Wide Web causes a disproportionate increase in client requests to highly accessed Web sites. The increasing demand, coupled with the growing importance of e-commerce, has precipitated the need for high performance server systems that are able to support million of accesses per day. While the first generation of Web sites was typically considered a channel for non critical information, where 90% of information was represented by static content, the second generation is characterized by a growing percentage of dynamically generated content, and represents the fundamental technology for information systems of the most advanced companies and organizations. The substantial changes transforming the Web from a communication and browsing infrastructure to a medium for conducting personal businesses, communications, and financial transactions are making the performance of Web-based services an increasingly critical issue. A common approach adopted by highly accessed Web sites is to rely on a multiple-server architecture that appears to users as a single virtual service [8,20]. An architecture of cluster-based Web servers (for short in this paper, *Web cluster*) refers to a set of server machines that are housed together in a single location, are interconnected through a high-speed network, and present a single system image to the outside. Each cluster node may be either a workstation, a PC or a symmetric multiprocessor. It usually contains its own disk and a complete operating system. Cluster nodes work collectively as a single computing resource. For example, massive parallel processing systems (e.g., SP-2), where each node satisfies all previous characteristics, can be considered a Web cluster.

Web clusters are often used for co-locating different Web sites; however, in this paper we consider a platform that hosts a single Web site. A Web cluster provides to the outside world a single virtual interface both at the site name level (e.g., `www.foo.com`) and at the IP level (e.g., 144.55.62.18). The only visible

address is a *Virtual IP* (VIP) address corresponding to a front-end node located in front of the set of servers. Thus, the authoritative DNS server for the Web site performs a one-to-one mapping by translating the site name into the VIP address. This solution makes the distribute nature of the cluster completely transparent to both users and client applications. The front-end node, hereafter called *Web switch*, acts as a centralized dispatcher with total control on assignment of client requests to servers. The switch receives all client packets destined to the VIP address and, through some routing mechanism, direct the packets to the Web server nodes that are selected by the dispatching policy running on the switch.

Cluster-based architectures can be broadly classified according to the OSI protocol stack layer at which the Web switch operates the request routing, that is *layer-4* (TCP level) or *layer-7* (application level) Web switches [20]. Most issues for Web switches operating at TCP/IP level have been solved, as demonstrated by the consistent number of products available on the market. On the other hand, we believe that Web switches working at application level need further investigation for what concerns both routing mechanisms and dispatching algorithms.

The focus of this paper is on dispatching algorithms for layer-7 Web switches, while some routing mechanisms are outlined in Section 2.2. These algorithms are referred to as *content-aware* because the requested content can be taken into account in making the assignment decision. The motivation for this study comes from the observation that most solutions either work only for specific classes of Web sites or achieve unsatisfactory performance when applied to more recent Web sites providing highly heterogeneous services. The contribution of this paper is twofold. We propose the first taxonomy of content-aware dispatching algorithms by distinguishing the type of information they use and their main objective. Moreover, we compare the performance of some representative content-aware algorithms for different workload scenarios.

Indeed, previous papers have compared content-aware policies against content-blind policies only (e.g., [7,17]), or they have focused on Web sites pro-

viding static content (e.g., [6]). On the other hand, in this paper we consider Web systems servicing heterogeneous content, such as static, dynamic, and secure information that impose different resource consumptions on the components of the Web cluster. Under workload characteristics that resemble those experienced by real Web sites, our simulation experiments demonstrate that dispatching algorithms aiming to load sharing can achieve much better performance than policies that aim to improve the reference locality or partition the servers nodes on the basis of the provided service. More specifically, our results indicate that the Content-Aware Policy proposed in [9], which classifies client requests on the basis of their expected impact on server components, is more effective than state-of-the-art dispatching policies when we consider Web sites providing static, dynamic, and secure services.

The remainder of this paper is organized as follows. In Section 2, we outline the typical architecture of a Web cluster and review some routing mechanisms. In Section 3, we discuss some dispatching algorithms for the Web switches operating at application level. In Section 4, we present a detailed simulation model for the Web cluster and the parameters of the workload model. In Section 5, we compare the simulation results of some representative content-aware algorithms under different workload scenarios. Finally, in Section 6 we present some concluding remarks.

2. Front-end Web switch

2.1. Classification

The Web switch plays a key role in a Web cluster. For this reason, we broadly classify the architecture alternatives according to the OSI protocol stack layer at which the Web switch operates the request routing. The main difference between a layer-4 and a layer-7 Web switch derives from the kind of information available to the Web switch when it takes routing decisions.

- Layer-4 Web switches perform *content-blind routing* (also referred to as *immediate binding*), because they determine the target server when the client establishes the TCP/IP connection, upon the arrival of the TCP SYN packet.
- Layer-7 Web switches can deploy *content-aware routing* (also referred to as *delayed binding*), by letting the switch establish a complete TCP connection with the client, examine the HTTP request, and then relay it to the target server.

Both layer-4 and layer-7 Web switches can be further classified on the basis of the data flow between the client and the target server. The main difference is in the return way server-to-client, because all client-to-server requests must flow through the Web switch. In *one-way* architectures, the target server responds directly to the client without passing through the switch. In *two-way* architectures, the server returns its response to the Web switch, which in its turn sends the response back to the client. In this paper we consider a one-way Web cluster that use a layer-7 Web switch (Figure 1), because this system offers higher scalability than two-way architectures.

Web cluster based on two-way architectures are provided by some research prototypes [1,9,11,22,23] as well as by some commercial products, such as Nortel Networks' Web OS SLB [16], Foundry Networks' ServerIron [13], Cisco's CSS [10], F5's BIG-IP [12], while one-way architectures have been deployed in [3,4,17,21] and up till now in one commercial product only [19].

2.2. Content-aware routing mechanisms

Content-aware routing requires more complex mechanisms than those for content-blind routing. The reason is that the HTTP request is first inspected before a decision is made about which server node should handle the request. To determine the request content, the Web switch must first establish a TCP connection with the client (i.e., the three-way handshake for the TCP connection

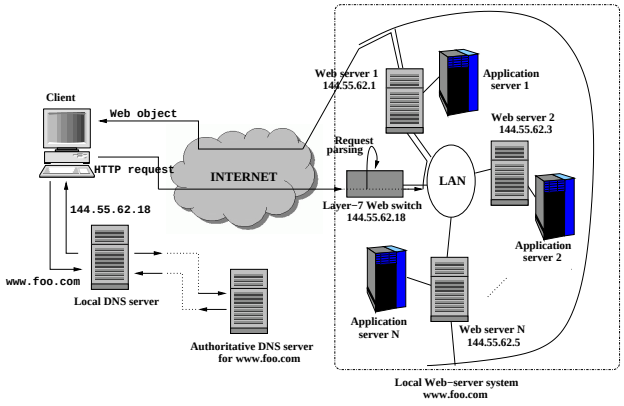

Figure 1. One-way Web cluster architecture with layer-7 switch.

establishment phase must be completed between the client and the Web switch) and then receive the HTTP request (i.e., the application level information). On the other hand, a layer-4 Web switch can choose the target server as soon as it receives the initial TCP SYN packet, before the client sends out the HTTP request.

Routing to the target server can be accomplished in one-way architectures by either a *TCP handoff* or a *TCP connection hop* mechanism. With TCP handoff, the Web switch establishes a TCP connection with the client, selects the target server, and then hands off its endpoint of the TCP connection to the server [3,17]. The handoff protocol is layered on top of TCP and runs on the Web switch and the servers, thus requiring changes to their operating systems. The handoff mechanism remains transparent to the client, as data packets sent by the servers appear to be coming from the Web switch and any acknowledgment packets sent by the client to the switch are forwarded to the target server by a module running at the bottom of the switch protocol stack.

TCP connection hop is a software-based proprietary solution developed by Resonate [19]. Once the Web switch has established the TCP connection with the client and selected the target server, it hops the TCP connection to the server. This is achieved by encapsulating the IP packet in an RPX packet and sending

it to the server [19]. Since the server shares the same VIP address, it can reply directly to the client. Acknowledgment packets and persistent session information from clients are managed by the Web switch.

When the Web switch uses HTTP/1.0, there is no difference between content-blind and content-aware routing for what concerns the granularity level on the client requests, because there is a one-to-one correspondence between HTTP requests and TCP connections. On the other hand, when the client/cluster interaction is based on HTTP/1.1 persistent connections, content-aware routing allows an assignment control with a granularity finer than that feasible to content-blind routing. Indeed, a layer-7 Web switch can assign requests traveling on the same TCP connection to different servers (e.g., servers 1 and 2 in Figure 2), thus achieving a granularity control down to individual HTTP requests. On the other hand, a layer-4 switch cannot assign the TCP connection to different servers, so multiple client requests traveling on the same persistent connection reaches the same server (e.g., server 1 in Figure 2).

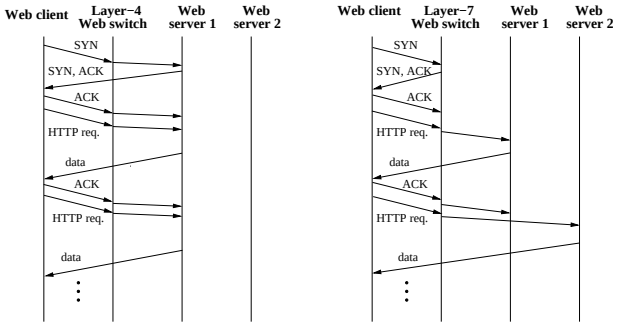

Figure 2. Possible request assignment of a layer-4 (left) and a layer-7 Web switch (right) when the HTTP/1.1 protocol is used.

However, layer-7 Web switches are not always preferable to layer-4 switches. The main drawback is that content-aware routing introduces a high processing overhead at the dispatching entity and may cause the Web switch to become

the system bottleneck. To augment cluster scalability, alternative designs for high performance Web clusters, which combine content-blind and content-aware request distribution, have been proposed in [4,21].

3. Content-aware dispatching algorithms

Layer-7 Web switches deploy content information aware distribution, as they can examine the entire HTTP request and dispatch it on the basis of detailed information about the client. Content-aware dispatching algorithms typically use either client information or a combination of client and server state information. We do not consider a layer-7 Web switch that do not use any state information or use server information only, because these kinds of algorithms could be implemented also in a layer-4 Web switch. A more expensive and sophisticated architecture such as a layer-7 Web switch is motivated only if its dispatching algorithm uses some client information. Therefore, we propose a simple taxonomy, where we first classify content-aware dispatching algorithms according to the information they use in making the dispatching decision.

Client information. A layer-7 Web switch can examine the entire HTTP request and can take decisions on the basis of detailed information about the client. The server selection can be based on the Web service/content requested, as URL content, SSL identifiers, and cookies.

Client and server information. In content-aware dispatching, the Web switch can also take into account some server state information, such as load condition, latency time, and availability as well as the current content of each server cache.

As a second level of classification, we consider the main goal of the dispatching policies. Indeed, information about the requested URL can be used

- to augment reference locality in the server caches so to reduce disk accesses (*cache affinity*),
- to use specialized server nodes to provide different Web services (*specialized servers*),
- to augment load sharing among the server nodes (*load sharing*).

Figure 3 summarizes the taxonomy of the content-aware dispatching policies and shows at the bottom level the policies that have been proposed and that we will describe below.

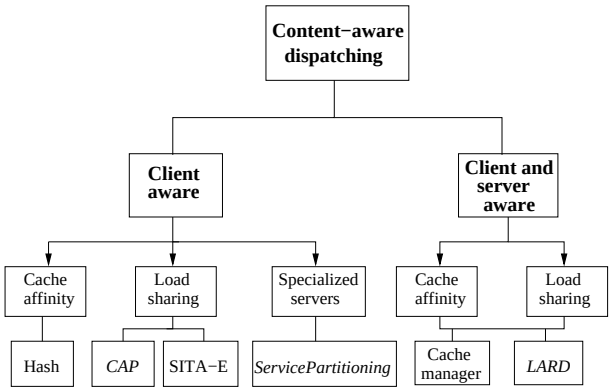

Figure 3. A taxonomy for content-aware dispatching algorithms.

3.1. Algorithms based on client information

Let us first consider the left part of the taxonomy in Figure 3. In cache affinity policies, the file space is typically partitioned among the server nodes. A hash function can be used to perform a static partitioning of the files. The dispatching policy running on the Web switch (namely, **Hash** algorithm) uses the same function. This scheme exploits at most the locality of references in the server nodes and achieves the best cache hit rate. However, it can be applied to Web sites providing static content only. Moreover, it ignores completely load

sharing, as it is almost impossible to partition the file space in such a way that the requests are balanced out. Indeed, if a small set of files account for a large fraction of requests (a well-know characteristic of Web workload, e.g., [2,18]), the server nodes containing those critical files will be more loaded than others.

For Web sites providing heterogeneous Web services, the requested URL can be used to statically partition the servers according to the service they handle. The goal is to employ specialized servers for different classes of requests, such as dynamic content, multimedia files, streaming video [22]. Most commercial content-aware switches deploy this type of approach (e.g., F5's BIG-IP [12], Resonate's Central Dispatch [19]). There are several possible choices for a a partition of the servers on the basis of the class of Web services. In this paper, as a representative example of these algorithms, we consider a policy (namely, **ServicePartitioning** algorithm) that partitions the services and the servers in three classes: for static, dynamic, and secure requests.

The third main goal of the content-aware dispatching algorithms is to improve load sharing among the server nodes. These strategies do not statically partition the file space nor the Web services. Two policies belong to this class: *Size Interval Task Assignment with Equal load* (SITA-E) and *Client-Aware Policy* (CAP). The former is more oriented to Web sites providing static information, the latter to Web sites providing heterogeneous services.

The **SITA-E** policy partitions dynamically Web content among the servers according to the file size distribution. It defines the size range associated with each server in such a way that the total load directed to each server is the same [14]. The Web switch determines the size of the requested file and selects the target server on the basis of this information. The goal is to assign short tasks to the lightly loaded nodes thus avoiding their mixing with heavy tasks. The SITA-E policy founds on the assumption that the service time of a request is proportional to its size; however, this assumption is valid for static information only.

The **CAP** policy takes into account the requested service as its goal is to improve load sharing in Web clusters that provide multiple services [9]. Indeed, most problems occur in load sharing when the Web site provides heterogeneous services that make an intensive use of different Web server resources. CAP policy classifies Web requests into four main categories on the basis of their impact on main Web server resources: static and lightly dynamic Web publishing services, disk-bound services, CPU-bound services, and disk- and CPU-bound services. Although the Web switch cannot estimate the service time of each client request accurately, it can distinguish the class of the request from the URL and estimate its impact on main Web server resources. The Web switch manages a circular list of server 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 server node is overloaded. CAP 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 dispatching choice is determined statically once the requested URLs have been classified.

Indeed, policies that use only client information have a great advantage over algorithms based also on server information. The latter policies often require expensive and hard to tuning mechanisms for monitoring and evaluating the load on each server, gathering the results, and combining them for dispatching.

3.2. Algorithms based on server and client information

Dispatching algorithms deployed at application level can also use a combination of client and server state information. In this section, we describe two policies that have been specifically designed to consider both client and server information; however, some client aware policies (e.g., CAP) can be easily supplemented with some server state information. Both proposed policies use client information for cache affinity purposes and server information for load sharing goals. For this reason, in the taxonomy in Figure 3, they belong to two classes.

The *Locality-Aware Request Distribution* (**LARD**) policy is a content-based request distribution that considers both locality and load sharing [3,17]. The basic principle of LARD is to direct all requests for the same object to the same server node until its utilization is below a given threshold. By so doing, the requested object is more likely to be found into the disk cache of the server node. The check on the server utilization is useful to avoid overloading servers and, indirectly, to improve load sharing. When a server utilization reaches a given watermark, the dispatcher assigns the request to a lowly loaded node, if it exists, or to the least loaded server. As we will see in the experimental results, the LARD policy gives best results for Web clusters that provide mainly static content. Its efficacy is reduced when the Web cluster provides static, dynamic, and secure content. A scheme similar to the LARD policy has been also implemented in the HACC cluster [23].

In the LARD policy, the Web switch maintains a mapping table from each requested file to the set of nodes that has served it. There is no feedback from the servers. On the other hand, the **Cache manager** dispatching policy relies on a cache manager that is aware of cache contents of all Web servers. Each server provides periodically this information to the cache manager. The Web switch selects the lightest loaded server having the object cached, provided that its load is below a certain threshold [7]. When the Web switch receives a request for an object that is not cached in any server, it selects the least loaded server.

In Section 5 we will compare the performance of the policies that are written in italic in Figure 3 for different workload scenarios.

4. Simulation model

In this section, we describe a detailed simulation model of the Web cluster shown in Figure 1 that we will use to compare the system performance for different content-aware dispatching algorithms executed by the Web switch.

4.1. System model

The Web cluster consists of multiple Web servers and application servers, and a dedicated front-end node that acts as a 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 application servers are private and invisible to the extern. Web servers, application servers, and Web switch are interconnected by a local fast Ethernet with 100 Mbps bandwidth. The Web cluster is connected to the Internet through one or more large bandwidth links that do not use the same Web switch connection to the Internet. Being the focus of this paper on Web cluster performance, we did not model the details of the external network that connects the clients to the Web cluster. The main components comprising the Web cluster are shown in Figure 1.

Each server in the cluster is modeled as a separate component with its CPU, main memory, locally attached disk, and network interface. The Web server software is modeled as an Apache-like server, where an HTTP daemon waits for connection requests.

The base HTML file and the embedded objects within the page are either retrieved from the disk (or cache) of the Web server, if they are static objects or generated by the application server, if they are dynamic objects. When a Web server needs a dynamic object to fulfill the client request, it generates a request to the application server. It is worth to observe that our model considers the overhead in dispatching requests at the layer-7 Web switch that are modeled with the values given in [3]. The simulation model has been implemented using the CSIM18 package [15]. Additional details regarding the system model can be found in [9].

4.2. Workload model

Special attention has been devoted to the workload model that incorporates recent results on the characteristics of contemporary Web traffic. The high variability and self-similar nature of Web access is modeled through heavy-tailed distributions such as Pareto, lognormal, and Weibull distributions [2,5,18]. Random variables generated by these distributions can assume extremely large values with non-negligible probability.

Client arrivals to the system follow an exponential distribution. The interarrival rate is set to 100 clients per second (cps), if not otherwise specified. The number of *requests* per client session that is, the number of consecutive page requests the client will submit to the Web site, is modeled according to the inverse Gaussian distribution [18]. The *user think time*, which represents the time between the retrieval of two successive Web pages from the same client, is modeled through a Pareto distribution [5]. The number of *embedded objects* per page including the base HTML page is also obtained from a Pareto distribution [5]. The distribution of the object sizes requested to a Web server is a hybrid function, where the body is modeled according to a lognormal distribution, and the tail according to a heavy-tailed Pareto distribution [2,5]. A summary of the distributions and parameters used in our simulation experiments is in Table 1.

To characterize the different Web services provided by the Web cluster, we consider three scenarios that impose different resource consumptions on the components of the Web cluster.

High cache hit rate. We consider a Web site providing *static* and *lightly dynamic* content. A static object resides on the disk of the Web server, it is modified with a relatively long time interval, and is always cacheable in the disk cache. A lightly dynamic object is cacheable with 0.3 probability. This workload scenario does not highlight any particular system bottleneck, because the cache hit rate is over 50%.

Table 1
Parameters of the system and workload model.

Web cluster	
Number of servers	8
Disk transfer rate	20 MBps
Memory transfer rate	100 MBps
HTTP protocol	1.1
Intra-cluster bandwidth	100 Mbps
Clients	
Arrival rate	10-100 clients per second
Requests per session	Inverse Gaussian ($\mu = 3.86$, $\lambda = 9.46$)
User think time	Pareto ($\alpha = 1.4$, $k = 2$)
Objects per page	Pareto ($\alpha = 1.33$, $k = 1$)
Object size (*body*)	Lognormal ($\mu = 7.640$, $\sigma = 1.705$)
Object size (*tail*)	Pareto ($\alpha = 1.383$, $k = 2924$)

Stress on Web servers. We consider a Web site providing static, dynamic, and secure services. In particular, 60% of requests are for lightly dynamic objects, 20% for secure objects, and 20% for dynamic objects generated by the application servers. These request percentages are chosen to stress the Web servers. The results of requests to applications servers are not cacheable; however, the limited percentage of dynamic requests do not stress the application servers. On the other hand, secure requests highly stress the CPU of the Web servers.

Stress on application servers. We consider a Web site providing heterogeneous services, where 40% of the requests are for lightly dynamic objects, 20% for secure document objects, and 40% for dynamically generated objects. Unlike the previous scenario, dynamic requests now use intensively the resources of the applications servers and their results are again not cacheable.

To model the generation of dynamic objects that occurs at the application server, we use a resource service time exponentially distributed with mean equal

to 300 msec. Our model of secure connections, which represent CPU-bound requests, includes all main CPU and transmission overheads due to SSL protocol interactions, such as key material negotiation, server authentication, and encryption and decryption of key material and Web information. More details regarding the secure workload model can be found in [9].

5. Experimental results

In this paper, we use the cumulative frequency of the *page latency time* as the main metric to analyze the performance of the Web cluster, because mean values for the latency time have a little meaning in this highly variable environment. The latency time measures the completion time of a page request at the Web cluster side. Being the focus of this paper on Web cluster performance, we do not include in the page latency time all delays related to the transmission over the Internet.

To analyze the impact of a dispatching policy on the load state of the main cluster components, such as Web server CPUs and disks or application servers, we also use the *Maximum Utilization* observed on the Web cluster components.

5.1. Impact of granularity of request dispatching

In this section, we make some performance considerations on the granularity of request dispatching. When the interaction between Web client and Web server is carried out through the HTTP/1.1 protocol, which allows the use of persistent connections, a content-aware Web switch may distribute the requests at two levels of granularity:

connection level (conn), in which the Web switch selects a target server on the basis of the content of the first client request that is, the HTML page request; successive requests belonging to the same page are routed to the previously assigned server.

object level (obj), in which the Web switch selects a target server on the basis of the content of each object request.

Dispatching at object level has the advantage of a full control on different types of Web requests with respect to the connection level. On the other hand, dispatching at connection level puts less overhead on the Web switch because only a fraction of requested URLs are analyzed. This result is confirmed by Figure 4 for a workload scenario where the highest stress is put on the Web switch. On the x-axis there is the mean client interarrival rate that is, the mean number of clients generated per second (cps). On the y-axis we have the maximum utilization observed in the simulation experiments for the main cluster components (Web switch, CPU of Web servers, application servers). Each utilization value is measured on an interval of 10 seconds. As we are interested to stress the Web switch component, we consider a workload scenario with high cache hit rate, in which lightly dynamic objects are cacheable with probability equal to 0.5 (greater than the default value of 0.3). Figure 4 demonstrates that the capacity of the Web switch is at 30 and 50 cps for object and connection granularity level, respectively. On the other hand, the utilizations of Web server CPU and application servers range between 0.2 and 0.45. Results shown in Figures 4 refer to the CAP dispatching policy; similar results were observed for the LARD algorithm.

When we pass to consider the page latency time, we see that the more fine-grained control of object level allows to improve the performance of all dispatching algorithms with respect to connection level dispatching. Figure 5 shows that in a scenario that does not stress the Web switch, the gain achievable by using object level dispatching ranges from 10% to 12.5%, independently of the dispatching algorithm. As in this paper we focus on system performance rather than on system scalability, in the following experiments request dispatching is done at object level.

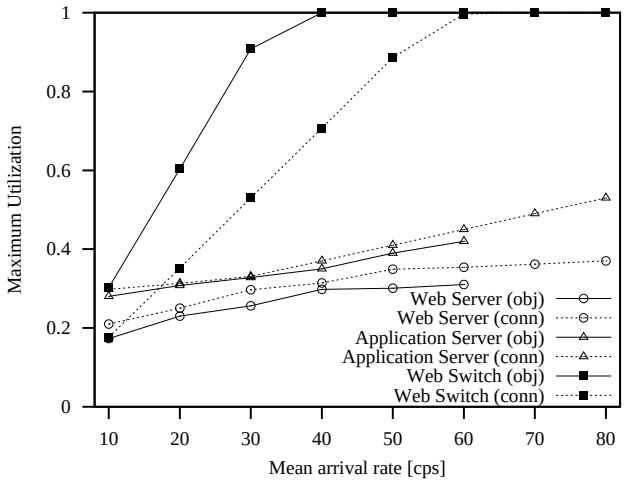

Figure 4. Maximum utilizations of Web cluster components for different granularities of request dispatching.

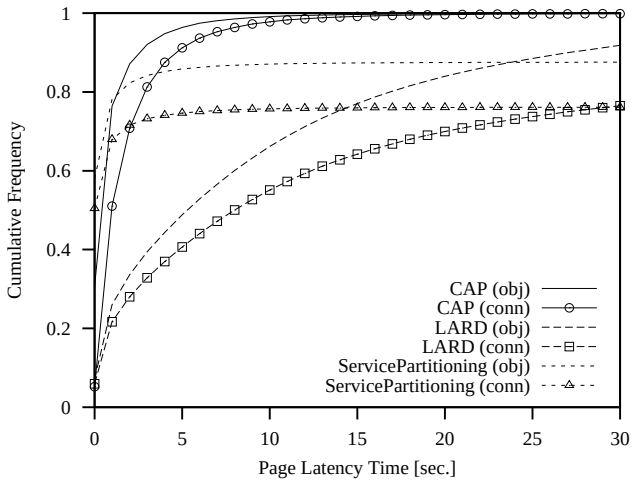

Figure 5. Page latency time for different granularities of request dispatching.

Let us now pass to compare the performance of the content-aware dispatching policies under the three workload scenarios described in Section 4.2: high cache hit rate, stress on Web servers, and stress on application servers.

5.2. High cache hit rate

In the scenario with high cache hit rate, the Web site provides static and lightly dynamic content. We consider the LARD and CAP policies only, because the request types are rather homogeneous and the ServicePartitioning has little meaning. Figure 6 shows that the LARD strategy, which exploits the reference locality of Web requests, performs slightly better than the CAP policy. In particular, LARD guarantees with very high probability that the page latency time is less than 1 second. The same latency time is guaranteed by CAP with probability equal to 0.9. The similar performance obtained by LARD and CAP polices is due to the absence of any overloaded cluster component. Indeed, if we look at the resource utilization of the main cluster resources, we observe that the maximum utilization is always less than 0.65. This means that neither Web server CPUs nor application servers are ever highly loaded.

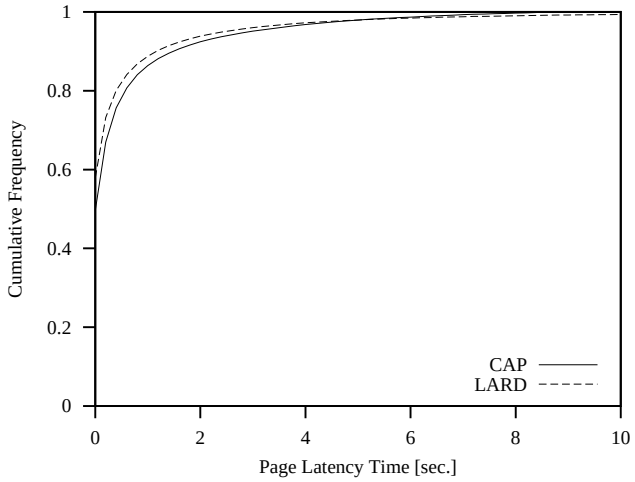

Figure 6. Page latency time for Scenario 1 (*high cache hit rate*).

5.3. Stress on Web servers

In this section, we compare the performance results achieved by the content-aware dispatching policies when the workload is CPU-bound. The improvement achieved by the CAP strategy is considerable, with respect to both LARD and ServicePartitioning. From Figure 7 we observe that CAP guarantees a page latency time of about 4 seconds with 0.95 probability, while for ServicePartitioning the same latency time is achieved with a probability of about 0.82, and for LARD with a probability of less than 0.5.

In particular, the ServicePartitioning is unable to guarantee any performance bound because some servers saturate their capacity and approximately 20% of clients requests do not receive any response. The latency time exceeds the connection timeout that in Unix systems is set to the default value of 120 seconds.

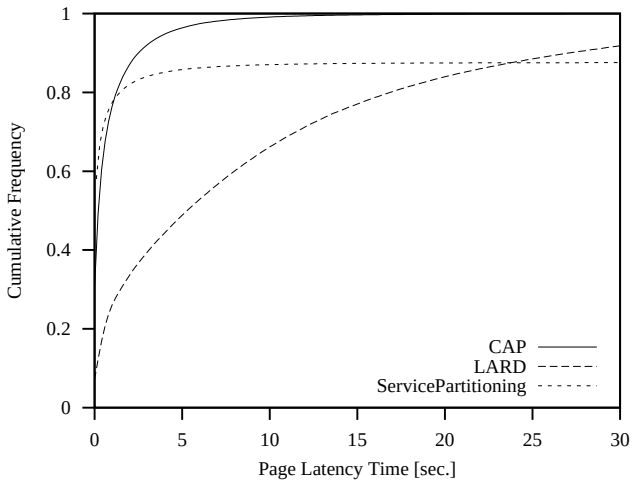

Figure 7. Page latency time for Scenario 2 (*stress on Web servers*).

In order to fully understand the behavior of the different policies, we analyze the CPU utilization of the Web servers. Figure 8 shows that for LARD and ServicePartitioning the CPU utilization exceeds 90%, meaning that some Web

server nodes are overloaded. This high level of utilization impacts negatively on the performance of the entire Web cluster. On the other hand, by applying the CAP policy, the maximum CPU utilization does not exceed 0.65. Indeed, as CAP assigns client requests reaching the Web cluster with the goal of sharing all classes of services among the servers, it is able to limit resource utilization.

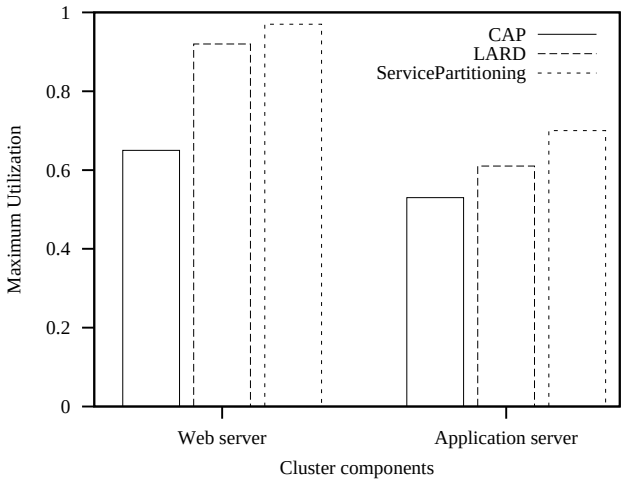

Figure 8. Maximum utilization of Web cluster components for Scenario 2 (*stress on Web servers*).

5.4. Stress on application servers

We evaluate now the performance of the content-aware dispatching algorithms under the scenario in which the stress is put on the application servers. To this purpose, we consider a Web site where 40% of requests need the intervention of an application server.

Figure 9 shows that the CAP policy still provides acceptable page latency times with high probability, while LARD and ServicePartitioning fail to guarantee any satisfactory performance. As an example, CAP policy achieves a page latency time of about 4 seconds with 0.90 probability. For ServicePartitioning, the same latency time is achieved with a probability of about 0.79, and for LARD

with a probability of less than 0.3. The peculiarity of LARD over ServicePartitioning is that there are servers that do not saturate even if highly utilized and continue to produce responses for clients. On the other hand, under the ServicePartitioning policy all servers dedicated to CPU- and application server-bound requests saturate and about 21% of client requests do not receive any response as the latency time exceeds all protocol timeouts.

This behavior is motivated by Figure 10 showing the maximum utilizations of Web servers and application servers. For both LARD and ServicePartitioning policies, the application server utilization exceeds 0.95, while no utilization is over 0.7 when CAP is employed.

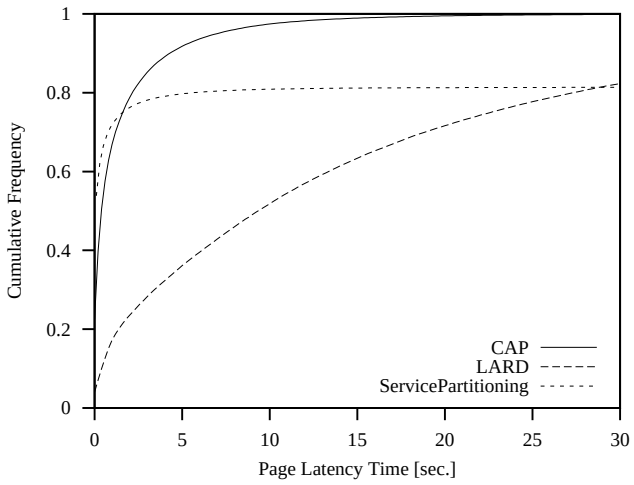

Figure 9. Page latency time for Scenario 3 (*stress on application servers*).

6. Conclusions

Web cluster architectures are becoming very popular for supporting highly accessed Web sites that offer heterogeneous services. In this paper, we consider Web switches that can use application level information to assign client requests to the target server node. We propose a taxonomy of the state-of-the-art content-

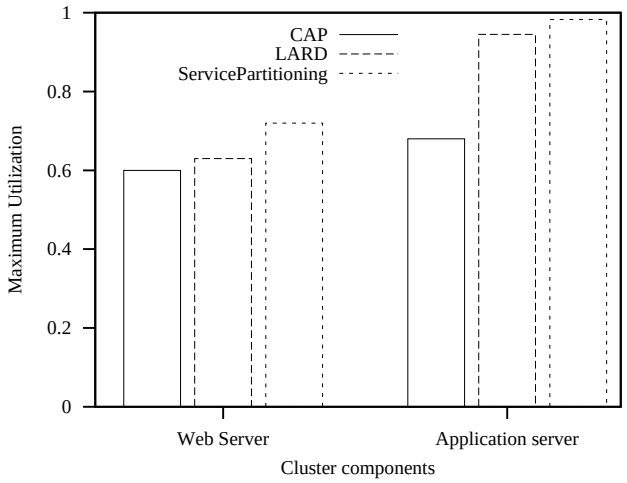

Figure 10. Maximum utilization of Web cluster components for Scenario 3 (*stress on application servers*).

aware dispatching algorithms on the basis of used state information (client, client and server) and main target (load sharing, reference locality, specialized servers). Moreover, we evaluate the performance of some representative policies for different workload scenarios. We demonstrate through a wide set of simulation experiments that dispatching policies that aim to improve hit rates in the server caches, such as LARD, give best results for Web sites providing static information and some simple database searches. On the other hand, when we consider Web clusters that provide highly heterogeneous content, only policies, such as CAP, that aim to share the load among cluster components, can provide satisfactory performance.

Acknowledgments

The authors acknowledge the support of Banca di Roma (Res. contract BDR-2001 on "Advanced technologies").

References

[1] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha, Design, implementation and performance of a content-based switch, in: *Proc. of IEEE Infocom 2000*, Tel-Aviv, Israel (March 2000).

[2] M. F. Arlitt and T. Jin, A workload characterization study of the 1998 World Cup Web site, *IEEE Network*, 14(3) (2000) 30-37.

[3] M. Aron, P. Druschel, and Z. Zwaenepoel, Efficient support for P-HTTP in cluster-based Web servers, in: *Proc. of USENIX 1999 Conf.*, Monterey, CA (June 1999).

[4] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers, in: *Proc. of USENIX 2000*, San Diego, CA (June 2000).

[5] P. Barford and M. E. Crovella, A performance evaluation of Hyper Text Transfer Protocols, in *Proc. of ACM Sigmetrics 1999*, Atlanta (May 1999) pp. 188-197.

[6] R. Bianchini and E. V. Carrera, Analytical and experimental evaluation of cluster-based network servers, *World Wide Web*, 3(4) (2000).

[7] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson, Achieving load balance and effective caching in clustered Web servers, in: *Proc. of 4th Int'l Web Caching Workshop*, San Diego, CA (April 1999) pp. 159-169.

[8] V. Cardellini, M. Colajanni, and P. S. Yu, Dynamic load balancing on Web-server systems, *IEEE Internet Computing*, 3(3) (1999) 28-39.

[9] E. Casalicchio and M. Colajanni. A client-aware dispatching algorithm for Web clusters providing multiple services, in: *Proc. of 10th Int'l World Wide Web Conf.*, Hong Kong (May 2001).

[10] Cisco Systems, http://www.cisco.com/.

[11] A. Cohen, S. Rangarajan, and H. Slye. On the performance of TCP splicing for URL-aware redirection, in: *Proc. of USENIX Symp. on Internet Technologies and Systems*, Boulder, CO (October 1999).

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

[13] Foundry Networks, http://www.foundrynet.com/.

[14] M. Harchol-Balter, M. E. Crovella, and C. D. Murta, On choosing a task assignment policy for a distributed server system, *J. of Parallel and Distributed Computing*, 59(2) (1999) 204-228.

[15] Mesquite Software Inc., CSIM18 user guide, http://www.mesquite.com/.

[16] Nortel Networks, Nortel Networks Web OS, http://www.nortelnetworks.com/.

[17] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, Locality-aware request distribution in cluster-based network servers, in: *Proc. of 8th ACM Conf. on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA (October 1998).

[18] J. E. Pitkow, Summary of WWW characterizations, *World Wide Web*, 2(1-2) (1999) 3-13.

[19] Resonate Inc., http://www.resonate.com/.

[20] T. Schroeder, S. Goddard, and B. Ramamurthy, Scalable Web server clustering technologies, *IEEE Network*, 14(3) (2000) 38-45.

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

[22] C. S. Yang and M. Y. Luo, A content placement and management system for cluster-based Web servers, in: *Proc. of 20th IEEE Int'l Conf. on Distributed Computing Systems*, Taipei, Taiwan (April 2000).

[23] X. Zhang, M. Barrientos, J. B. Chen, and M. Seltzer, HACC: An architecture for cluster-based Web servers, in: *Proc. of 3rd USENIX Windows NT Symp.*, Seattle, WA (July 1999).