Mechanisms for Quality of Service in Web Clusters

Valeria Cardellini, Emiliano Casalicchio
University of Roma Tor Vergata
Roma, Italy 00133
{cardellini, ecasalicchio}@ing.uniroma2.it

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

Salvatore Tucci
University of Roma Tor Vergata
Roma, Italy 00133
tucci@uniroma2.it

Abstract

The new generation of Web systems provides more complex services than those related to Web publishing sites. Users are increasingly reliant on the Web for up-to-date personal and business information and services. Web architectures able to guarantee the *Quality of Service* (QoS) that rules the relationship between users and Web service providers require a large investment in new algorithms and systems for dispatching, load balancing, and information consistency. In this paper, we consider Web cluster architectures composed of multiple back-end server nodes and one front-end dispatcher and we analyze how to provide differentiated service levels to various classes of users. We demonstrate through simulation experiments under realistic workload models that the proposed mechanisms are able to satisfy QoS requirements of the most valuable users classes, without impacting too negatively on the other users.

Keywords: Distributed systems, Quality of Service, Load sharing, Performance evaluation.

1 Introduction

The Web is becoming an important channel for critical information and the fundamental technology for information systems of the most advanced companies and organizations. Users are becoming increasingly reliant on the Web for up-to-date personal, professional, and business information. The substantial changes transforming the Web from a communication and browsing infrastructure to a medium for conducting personal businesses and e-commerce are making quality of Web service an increasingly critical issue. This new scenario encourages the design and implementations mechanisms and algorithms able to guarantee the Quality of Service (QoS) that will rule the relationship between users and Web services providers.

Because of the complexity of Web infrastructure, many factors affect the Web performance. Hence, to guarantee the assessed service levels for QoS parameters, modifications on all components of the Web would be required: from network technology and protocols, to hardware and software architectures of Web servers and proxies. Many efforts in the field of QoS focus on solution at the network level [6, 7]. However, network QoS by itself is not sufficient to support end-to-end QoS. Furthermore, with the network bandwidth increasing much faster than the server capacity, it is more likely that the bottleneck will be on the server side. To avoid that high priority network traffic being dropped at the server, the Web servers should have mechanisms and policies for delivering end-to-end QoS. This requires the definition of classes of services, choice of priority levels, guarantee of different QoS requirements through priority scheduling disciplines, and

monitors for starvation of low priority services [5]. Moreover, to provide stable service to preferred users, it is necessary that the system can be able overloaded conditions gracefully.

When the number of accesses grows exponentially, systems with multiple server nodes are leading architectures for building Web sites that have to guarantee QoS. In this paper we consider as basic architecture a cluster-based Web server (namely, *Web cluster*). From the client's point of view, any HTTP request to a Web cluster is presented to a front-end server that acts as a representative for the Web site. This component, called a *Web switch*, retains transparency of the distributed architecture for the client and distributes all incoming requests to the Web server nodes. In this paper, we propose and compare various mechanisms that add QoS attributes to a Web cluster architecture.

Most projects about quality of Web services propose new Web server architectures that support differentiated services thereby enabling preferential treatment to some classes of service [5, 14, 16]. The main components of a Web server architecture able to support differentiated services include a *classification* mechanism to assign different *service classes* to incoming requests, an *admission control* policy to decide how and when to reject requests according to the service class they belong to, a performance isolation mechanism that can be implemented through a *request scheduling* policy that decides the order in which requests should be served, and a *resource scheduling* policy that decides how to assign system resources to different service classes. Admission control has been also proposed as a first key mechanism to prevent performance degradation of Web services [10].

The second key mechanism that has significantly improved QoS in Web servers is the concept of *performance isolation*. This ensures that each service class receives a certain share of server resources (i.e., CPU, memory, disk, network bandwidth) [3]. An algorithm that combines classification and performance isolation mechanisms has been recently proposed in [12]. To meet the QoS expected by different classes of services, the most of the investigated solutions require the introduction of new priority scheduling disciplines [5, 14, 16] at the Web server level. In this paper, we study some mechanisms for a QoS-enabled Web cluster by acting on the requests dispatching policy carried out by the Web switch and evaluate their impact on user response time. We also investigate the combination of a priority-based server scheduling discipline with Web switch dispatching policies.

The remainder of the paper is organized as follows. In Section 2, we outline the typical architecture for a Web cluster. In Section 3, we propose some global dispatching algorithms for the Web switch that take into account user differentiation and service level provision. In Section 4, we outline the simulation model and discuss experimental results. In Section 5, we summarize the main conclusions of this paper.

2 Multi-node Web systems

Web systems with multiple nodes are leading architectures for building popular Web sites that have to guarantee scalable services and support ever increasing request load. The considerable number of academic and commercial proposals reveals the continuously growing interest in distributed Web architectures. In this paper, we focus on locally distributed Web systems, namely *Web clusters*, that have been classified in [9, 15]. 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 is publicized with one site name to provide a single interface to users. To control all of 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 (*VIP*) that corresponds to the address of a *Web switch*. The authoritative DNS server for the Web site translates the site name into the IP address of the Web switch. In such a way, the Web switch acts as a centralized global dispatcher that receives the totality of the requests and routes them among the servers of the cluster.

Web clusters can provide fine-grained control on request assignment, high availability, and good scalability. The architecture alternatives can be broadly classified according to the OSI protocol stack layer at which the Web switch operates the request assignment, which is *layer-4* and *layer-7* Web switches. The main difference is the kind of information available at the Web switch to perform assignment and routing decision [15]. In this paper, we focus on Web switches operating at layer-4 that is, at TCP/IP level. Therefore, the type of information regarding the client is limited to that contained in TCP/IP packets that is, IP source address, TCP port numbers, and SYN/FIN flags in the TCP header. Since packets pertaining to the same TCP connection must be assigned to the same Web server node, the client assignment is managed at TCP session level. The Web switch maintains 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 bits in the flag field determines if the packet pertains to either a new or an existing connection.

Layer-4 Web switches can be further classified on the basis of the mechanism used by the Web switch to route inbound packets to the target server and the packet path between the client and server. In this paper, we consider a one-way architecture where the routing to the target server is accomplished by forwarding the packet at MAC level and only inbound packets flow through the Web switch [11]. This architecture reduces the delay introduced by the switch in the dispatching process and prevents the bridge(s) to the external network becoming a potential bottleneck for the Web cluster throughput. Figure 1 shows the main components of the Web cluster architecture we consider in this paper. Multiple Web servers manage static requests, while back-end servers are dedicated to dynamic requests. The QoS admission control mechanism, if present, is integrated into the Web switch.

3 Web switch dispatching policies

In this section, we review traditional dispatching algorithms that are currently used by most layer-4 Web switches and propose new dispatching policies with the goal of providing QoS in term of response latencies by taking into account some priority related information. We refer to the first class of dispatching policies as *QoS-blind*, while the latter is called *QoS-aware*. Since each Web server in the cluster can be enhanced to support QoS by acting on the server resources scheduling, we also explore the combination of Web switch dispatching policies with a QoS-enabled Web server. In the following, we refer to *service class* to denote the differentiation of incoming requests into classes. The proposed policies offer support to multiple service classes. Specifically, in this paper we consider, without loss of generality, three classes of service levels, denoted as *high*, *medium*, and *low* service level.

The Web switch may use various global dispatching policies to assign the load to the nodes of a Web cluster. The main alternative remains the kind of system information used for the assignment decision. *Static* policies do not consider any system state information. *Dynamic* policies use some system state information that may be either some *server state information*, such as load condition, response time, availability, or network utilization, some *client information*, whose availability depends on the level at which the switch operates, or even a combination of client and server information.

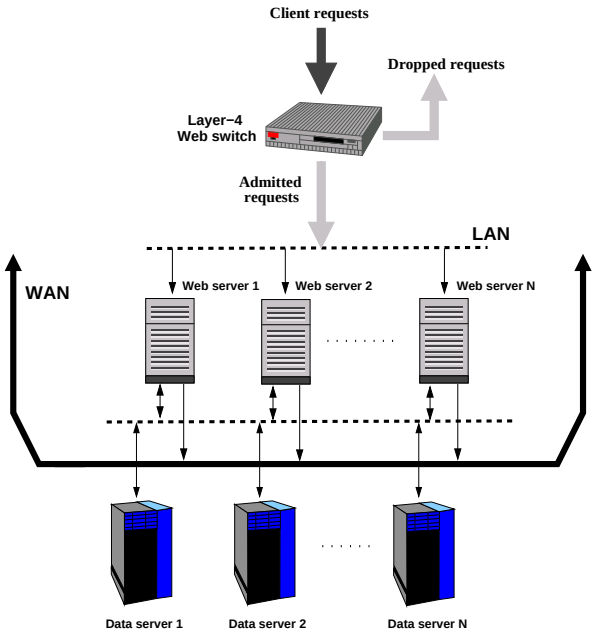

Figure 1: Web cluster architecture.

Dynamic algorithms have the potential to outperform static algorithms by using some state information to help dispatching decisions. Typical server load indexes include the CPU utilization evaluated over a short interval, the instantaneous CPU queue length periodically observed, and the instantaneous number of active connections periodically measured on each server. We select as a load index the number of active connections that in various implementations of Web clusters has been proved to be the most effective measure [15].

Since our architecture uses a layer-4 Web switch, the granularity of the dispatched object depends on the HTTP version that is used in client-server interaction. The difference is the number of HTTP requests that flow in a single TCP connection. In HTTP/1.0, there is a one-to-one correspondence between HTTP request and TCP connection. On the other hand, since HTTP/1.1 allows requests pipelining, multiple HTTP requests can use the same TCP connection. As this flow does not allow the Web switch to differentiate its assignment to different servers, the request granularity on which the assignment is activated can be at object request level in HTTP/1.0, while at connection request level in HTTP/1.1. In this paper, we assume that both clients and servers support HTTP/1.1, so we use the term of request instead of connection request when referring to the assignment decision.

3.1 QoS-blind dispatching policies

We first consider dispatching policies that do not provide differentiated quality of service to incoming requests. Two widely used static dispatching policies are *random* and *round-robin* (**RR**) algorithms. A simple dynamic policy using server load information is *least loaded server* (**LL**), where each request is assigned to the lowest loaded server measured in the last observation interval. However, *weighted round-robin* (**WRR**) is one of the best performing policies for Web clusters. It is widely adopted in real Web clusters [11] and all experimental and simulation results carried out by our group in the last years verified that this policy represents the best

compromise between simplicity and efficacy. WRR comes as a variation of the round robin policy. WRR assigns to each server a dynamically evaluated weight that is related to the server load state. The Web switch periodically gathers this information from servers and computes the weights, that are dynamically incremented for each new connection assignment.

In a QoS-blind Web switch, each Web server can be enabled with some mechanism for QoS support, as proposed in literature [1, 5, 16]. Specifically, the Web switch determines the target server using a QoS-blind dispatching policy, and the Web server adopts a priority scheduling of its own resources, after having determined the service class of the request. We refer to the **SerADM-SerPRI** policy if the server prioritizes the scheduling of its resources on the basis of the service class, after having identified requests to be dropped through an admission control mechanism. In particular, the service can be denied only to requests belonging to the low class. The admission control mechanism is triggered independently by each Web server when its load exceeds a threshold. This is set as a percentage of the maximum number of concurrent active connections $MaxConn$ that the server can sustain without degrading performance. Every T_{get} seconds, each Web server calculates its load, and determines if the admission control has to be switched on or off. The rejection phase starts when the server load exceeds the threshold Thr and ends when the load returns under the same threshold value. We set $Thr = \alpha \cdot MaxConn$, where $0 < \alpha \leq 1$ allows to augment or diminish the percentage of dropped requests.

3.2 QoS-aware dispatching policies

In this section, we propose some dispatching policies that achieve quality of Web services by acting on the assignment decision carried out by the Web switch. All the policies we discuss are based on the principle that the first mechanism to guarantee a given QoS is to control the load level on each server in the cluster. This can be easily achieved by acting on the maximum number of connections that each server can establish. Hence, to satisfy the performance target for the most demanding class and prevent it experiencing performance degradation due to an overloaded cluster, we implement an admission control that is integrated into the Web switch dispatching policy. When the admission control policy is switched on, access to Web services can be denied to the requests belonging to the lowest class.

The proposed QoS-aware dispatching policies can be grouped on the basis of the set of servers being assigned to each service class: *QoS-aware dispatching without server partition* and *QoS-aware dispatching with server partition*. The admission control mechanism is triggered by the Web switch on the basis of the servers' load. Once the request has been admitted to the system, the Web switch can use a static (e.g., RR) or dynamic (e.g., LL or WRR) policy to select the target server in the corresponding set. This can be any server of the cluster if no other partition is used or a subset of the servers otherwise.

In *QoS-aware dispatching without server partition*, each request can be assigned to any server, independent of the service class it belongs to. The admission control mechanism is triggered by the Web switch when $SumLoad$, that is the current sum of the servers load, exceeds a threshold Thr, which is set to $\alpha \cdot N \cdot MaxConn$, where $0 < \alpha \leq 1$, and N is the number of Web servers. Every T_{get} seconds, the Web switch gathers the number of active connections from each server, calculates $SumLoad$, and determines if the admission control has to be switched on or off. The rejection phase starts when $SumLoad$ exceeds the threshold Thr and ends when it returns under the same threshold value. Only requests belonging to the low class can be refused. We refer to this policy as **SwiADM**.

We also propose an **SwiADM-SerPRI** algorithm that combines the SwiADM policy with some priority

scheduling discipline implemented at the Web servers. This policy differs from SerADM-SerPRI in that in the latter the admission control is carried out by each Web server and not by the Web switch.

In *QoS-aware dispatching with server partition*, the servers in the Web cluster are partitioned into two sets, denoted as *High Set* (HS) and *Low Set* (LS), respectively. Incoming requests classified as high are assigned to servers in the first set, while servers in LS serve requests belonging to medium and low classes. The admission control policy can reject requests belonging to the lowest class.

The server partition into two sets is dynamically determined that is, the cardinality of each set varies depending on server load. Let us assume that there are N servers in the cluster, at a given time t, it results $HS(t) = \{1, \ldots, us(t)\}$ and $LS(t) = \{us(t)+1, \ldots, N\}$, where $HS(t) \cap LS(t) = \emptyset$, and $us(t) \in \{1, \ldots, N\}$.

The upper limit of HS, that is $us(t)$, is initially set to $\lceil \rho N \rceil$, where ρ represents the percentage of connection requests belonging to the high class. Then, $us(t)$ is periodically evaluated when new load measures are received by the Web switch. If $SumLoad_{HS}(t) > \alpha \cdot us(t-1) \cdot MaxConn$, where $SumLoad_{HS}(t)$ denotes the sum of the servers load in $HS(t-1)$ and $0 < \alpha \leq 1$, then a new server is added to $HS(t)$ that is, $us(t) = us(t-1) + 1$. From that point, the moved server will receive only new requests from the high class, while it will continue to serve requests on already open connections with medium and low classes. The moved server will return to $LS(t)$ that is, $us(t) = us(t-1) - 1$ when $SumLoad_{HS}(t) < \alpha \cdot us(t-1) \cdot MaxConn$. Only requests from the lowest class can be refused when $SumLoad_{LS}(t) > (N - us(t)) \cdot MaxConn$. We refer to this policy as the **SwiADM-SerPART** algorithm.

4 Performance analysis

4.1 Simulation model

The focus of our study on Web cluster performance allows us to avoid the details of network architecture and Internet traffic. On the other hand, we model the details of all cluster components, the client-server interactions (based on an open system model), and the workload offered to the cluster. Each server is modeled as a separate component with its own 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. When the admission control is switched on, a connection request may be rejected. If it happens, the client resends the request after a timeout. If the request is still rejected, it is aborted. The simulation model is implemented using the CSIM18 package [13]. Details regarding the parameters of the system model can be found in [8].

Special attention has been devoted to the workload model that incorporates the most recent results on Web load characterization. The high variability and self-similar nature of Web access load is modeled through heavy-tailed distributions such as Pareto and lognormal distributions [2, 4]. In this paper, we consider two main classes of services provided by the Web site that is, *static Web service* composed by requests for HTML pages with some embedded static objects, and *dynamic Web service*, where some objects belonging to the page are dynamically generated through Web server and back-end server interaction. Our workload composition is given by 85% of static requests and 15% of dynamic requests. Those are further classified as high, medium, and low intensive, on the basis of the impact they impose on the back-end nodes; their relative percentage is set to 85%, 10% and 5%, respectively. The distribution functions and parameter values we use in the workload model can be found in [8].

4.2 Experimental results

The main objective of this study is to understand the impact that different QoS-enabled policies applied to the Web switch and/or the Web server have on user response time. As performance metrics, we use the *percentage of dropped requests* and the *90-percentile of page delay* that measures the completion time of a Web page at the server side (we do not consider the network impact). An efficient QoS-enabled policy should be able to guarantee adequate performance to the high service class without impacting too negatively on the performance of the other classes. From the selected performance metrics, it follows that a tradeoff between the 90-percentile of page delay and the percentage of refused connections should be established.

In our experiments, we compare the performance of QoS-enabled policies with that of the best QoS-blind policy. Figure 2 shows the cumulative distribution of the page delay for RR, WRR, and LL dispatching policies. Since WRR policy clearly outperforms all other algorithms, in the following experiments we consider WRR as the dispatching policy to assign admitted connection requests to a target Web server. WRR can be integrated or not with some QoS mechanisms.

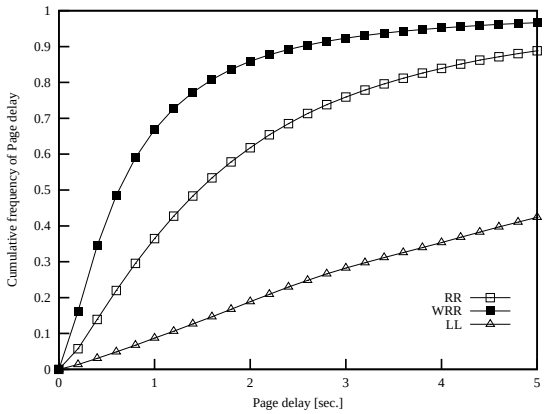

Figure 2: Cumulative frequency of page delay for QoS-blind dispatching algorithms.

In this paper, we consider three service classes that is, high, medium, and low class. In the first set of experiments, their percentage is set to 30%, 26.5%, and 43.5%, respectively. In Figure 3 we compare the performance of the proposed QoS-enabled policies (SwiADM, SwiADM-SerPRI, SerADM-SerPRI, and SwiADM-SerPART) and that of WRR. We show the 90-percentile of page delay for all service classes. We observe that all QoS-enabled policies outperform WRR: the performance gain reaches about 30% for high class requests. However, the main problem from the QoS point of view is that SwiADM and WRR do not differentiate services, thus achieving the same page delay for all service classes. On the other hand, SwiADM-SerPRI, SerADM-SerPRI, and SwiADM-SerPART policies that integrate an admission control with some performance isolation mechanism are able to guarantee different performance to different classes of services. Basically, these three policies obtain a similar page delay for both the high and medium classes, while the main difference exist for the low service class. SerADM-SerPRI and SwiADM-SerPART outperform SwiADM-SerPRI with a gain of about 30% on the 90-percentile of page delay. However, if we consider the percentage of dropped requests shown in Figure 4, we see that SwiADM-SerPRI rejects about 0.7% of low class requests. This is less than 30% of the requests dropped by SerADM-SerPRI and greatly less than the requests dropped by SwiADM-SerPART. Indeed, SwiADM-SerPART policy is able to differentiate the page

delay for each service class, but at the price of a much higher percentage of dropped requests.

SwiADM-SerPRI is better than SerADM-SerPRI if we consider the tradeoff between the 90-percentile of page delay for all classes and the percentage of dropped requests. Furthermore, it does not introduce any overhead on Web servers because the admission control is carried out at the dispatcher level. On the other hand, SerADM-SerPRI and all other policies that implement the admission control at the server level pay for the overhead of the connection that is first established and then refused at the server level.

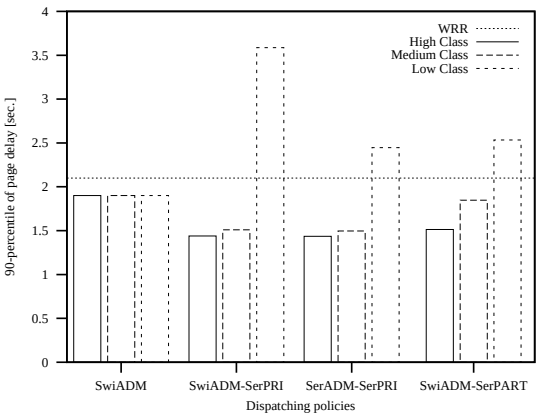

Figure 3: 90-percentile of page delay.

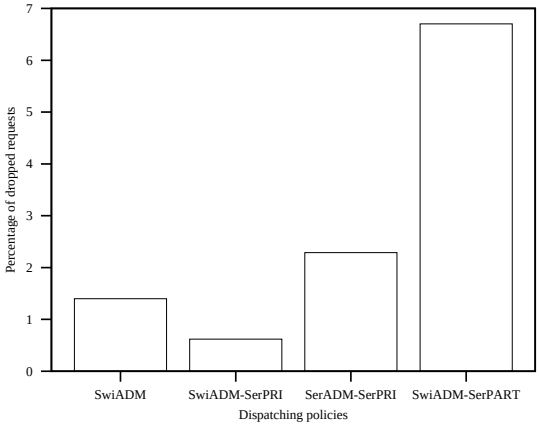

Figure 4: Percentage of dropped requests.

4.3 Sensitivity analysis

In the second set of experiments, we analyze the behavior of the dispatching policies when the composition of request classes changes. Specifically, we analyze the sensitivity to the percentage of high class requests that in our experiments ranges from 0.2 to 0.8. To reduce the degrees of freedom, in the following figures we set the ratio between medium and low class requests to 3/5. Other not reported experiments with different ratios do not modify the main conclusions of this paper.

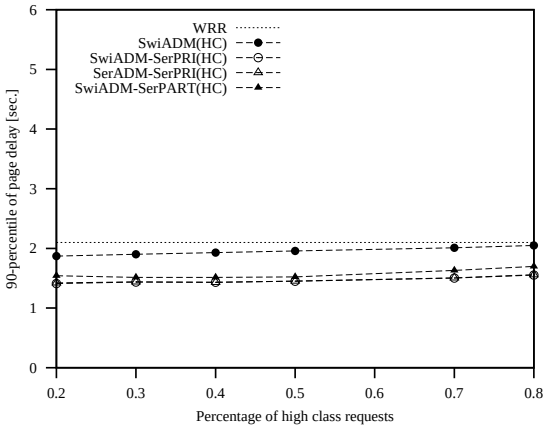

Figure 5: Sensitivity to high class percentage: 90-percentile of page delay for *high class* (HC).

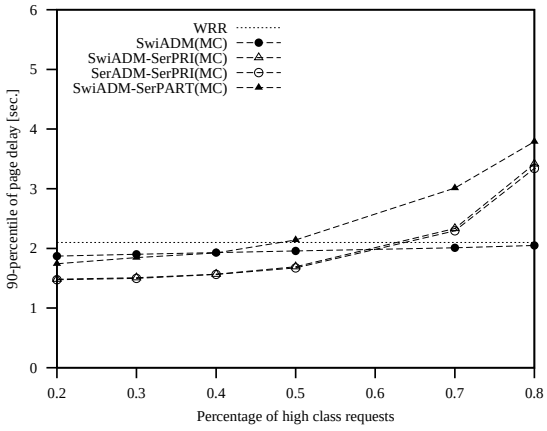

Figure 6: Sensitivity to high class percentage: 90-percentile of page delay for *medium class* (MC).

Figures 5, 6, and 7 show the trend of 90-percentile of page delay achieved by all presented policies for high, medium, and low class services, respectively. If we look at the high class (Figure 5), all QoS-enabled policies demonstrate quite stable results as a function of different service class compositions: indeed, the page delay has a very little range from 1.49 to 1.69 seconds. The robustness of QoS-enabled policies with respect to high class requests is an important result because during a day a Web site can be subject to very different load compositions in the reality.

For medium class requests (Figure 6), the page delay increases for higher percentages of high class requests. This was expected because requests belonging to the medium class are never dropped and therefore their delay time is highly sensitive to an augmented percentage of high class requests. In particular, the 90-percentile of page delay ranges from 1.74 to 3.78 seconds for SwiADM-SerPART policy, while it ranges from 1.47 to 3.41 seconds for priority-based policies. Therefore, from Figure 6 we can conclude that priority-based policies work better than SwiADM-SerPART policy for medium class requests.

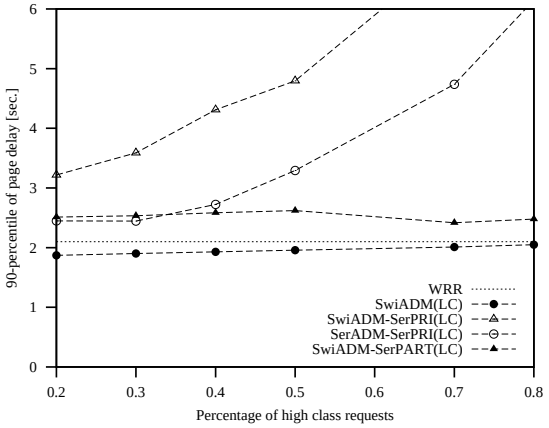

Figure 7: Sensitivity to high class percentage: 90-percentile of page delay for *low class* (LC).

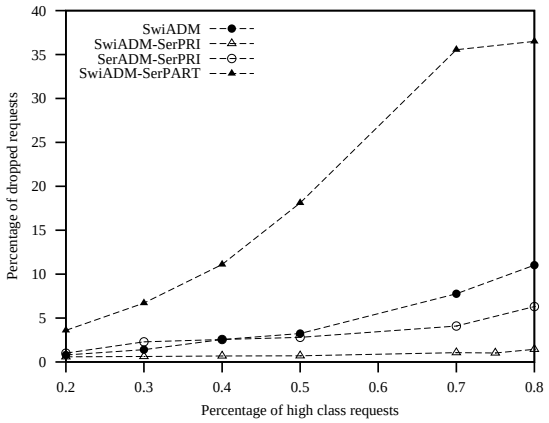

Figure 8: Sensitivity to high class percentage: percentage of dropped (*low class*) requests.

For low service class requests, the policies behave quite differently, as shown in Figure 7. SwiADM-SerPART strategy appears to be more stable than priority-based policies, at the price of a higher percentage of dropped requests (see Figure 8). Indeed, SwiADM-SerPART guarantees to low class requests a page delay of less than 2.5 seconds for any percentage of high class requests, while SwiADM-SerPRI and SerADM-SerPRI range from 3.2 to 9.6 seconds and from 2.44 to 6.14 seconds, respectively.

Figure 8 compares the percentage of low class dropped requests for QoS-aware policies. It shows the higher price paid by SwiADM-SerPART policy to guarantee a stable page delay for low class requests. While SwiADM-SerPART rejects from 3.6% to 30.6% of low class requests, priority-based policies deny services to less than 5% of requests. Therefore, even from this point of view, the best policy appears to be the SwiADM-SerPRI, because it refuses less than 1% of requests and obtains a 90-percentile of page delay for low class requests always below 10 seconds.

This second set of experiments confirms the main conclusions of the previous results. In particular, we can conclude that an admission control mechanism at the Web switch or at the Web server level provides better performance than QoS-blind policy (for example, the reader can compare SwiADM with WRR). However, admission control is not sufficient to differentiate service performance. We need to integrate it with some

performance isolation mechanism. This can be done at the level of server resources (e.g., SerPRI) or at the level of cluster resources (e.g., SerPART). These types of policies achieve all main QoS goals. Performance of SwiADM-SerPRI, SerADM-SerPRI and SwiADM-SerPART policies do not differ much for the high classes of services that are the most important from the QoS point of view. Substantial differences exist at the level of low classes of requests. SwiADM-SerPART guarantees better performance at the price of a large number of dropped requests. SwiADM-SerPRI provides opposite results. The choice between these two mechanisms depends also on preferences and choices that are in charge of the Web site management. The important result is that, once decided the Web site policy, we can use a combination of mechanisms to enforce it.

5 Conclusions

In this paper we have described and evaluated some mechanisms that can be integrated into existing Web cluster architectures to support quality of Web services. Using simulation experiments, we have found the admission control mechanism implemented either at the Web switch or at Web server is necessary to support quality of service in Web clusters. However, to provide differentiate service performance and satisfy the basic requirements for quality of Web services, the admission control mechanism has to be integrated with some performance isolation mechanism, such as priority scheduling of server resources or servers dynamic partitioning. Specifically, our simulations experiments show that policies based on admission control and server partition techniques achieve good and stable performance for low class of requests with the drawback of a large percentage of dropped requests. On the other hand, SwiADM-SerPRI policy, which implements the performance isolation through priority scheduling at the Web servers, allows the drop of a percentage of requests that is insensible to the frequency of high class requests. This result is obtained at the price of a page delay that increases greatly for larger percentages of high class requests.

In our future work we will implement the QoS-aware dispatching policies in a Web cluster based on a Linux/Apache environment. However, the real challenge remains the combination of network and server QoS mechanisms so to achieve an end-to-end quality of Web services.

References

[1] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated levels of service in Web content hosting. In *Proc. of Workshop on Internet Server Performance*, Madison, WI, June 1998.

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

[3] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proc. of ACM Sigmetrics 2000*, pages 90–101, Santa Clara, CA, June 2000.

[4] P. Barford and M. E. Crovella. A performance evaluation of Hyper Text Transfer Protocols. In *Proc. of ACM Sigmetrics 1999*, pages 188–197, Atlanta, May 1999.

[5] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, Sept./Oct. 1999.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An architecture for differentiated services.* RFC 2475, Dec. 1998.

[7] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: An overview.* RFC 1633, June 1994.

[8] V. Cardellini, E. Casalicchio, and M. Colajanni. A performance study of distributed architectures for the quality of Web services. In *Proc. of Hawaii Int'l Conf. on System Sciences (HICSS-34)*, Maui, Hawaii, Jan. 2001.

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

[10] L. Cherkasova and P. Phaal. Session based admission control: A mechanism for improving performance of commercial Web sites. In *Proc. of Int'l Workshop on Quality of Service*, London, UK, June 1999.

[11] G. S. Hunt, G. D. H. Goldszmidt, R. P. King, and R. Mukherjee. Network Dispatcher: A connection router for scalable Internet services. *Computer Networks*, 30(1-7):347–357, 1998.

[12] V. Kanodia and E. W. Knightly. Multi-class latency-bounded Web services. In *Proc. of Int'l Workshop on Quality of Service*, Pittsburgh, PA, June 2000.

[13] Mesquite Software Inc. *CSIM18 user guide*. Austin, TX. `http://www.mesquite.com/`.

[14] R. Pandey, J. F. Barnes, and R. Olsson. Supporting quality of service in HTTP servers. In *Proc. of ACM Symp. on Principles of Distributed Computing*, pages 247–256, Puerto Vallarta, Mexico, June 1998.

[15] T. Schroeder, S. Goddard, and B. Ramamurthy. Scalable Web server clustering technologies. *IEEE Network*, 14(3):38–45, May/June 2000.

[16] N. Vasiliou and H. L. Lutfiyya. Providing a differentiated quality of service in a World Wide Web server. *ACM Performance Evaluation Review*, 28(2):22–28, Sept. 2000.

Vitae

Valeria Cardellini received a Ph.D. in computer engineering at University of Roma Tor Vergata in June 2001. She received the Laurea degree (master) in computer engineering from the University of Roma Tor Vergata in 1997. In 1999 she spent six months at IBM T.J. Watson Research Center as a visiting researcher. Her research interests focus on the areas of modeling and simulation with particular emphasis on distributed systems and algorithms for the World Wide Web. Valeria Cardellini is a member of the IEEE Computer Society and the ACM.

Emiliano Casalicchio is currently pursuing a Ph.D. in computer engineering at University of Roma Tor Vergata. He received the Laurea degree (master) in computer engineering from the University of Roma Tor Vergata in 1998. His research interests focus on the areas of performance analysis, parallel architectures, and technologies with particular emphasis on Web architectures. Emiliano Casalicchio is a member of the IEEE Computer Society.

Michele Colajanni is currently a full professor in the Department of Computer Engineering at the University of Modena, Italy. He received the Laurea degree (master) in computer science from the University of Pisa in 1987, and the Ph.D. degree in computer engineering from the University of Roma Tor Vergata in 1991. From 1992 to 1998, he was at the University of Roma Tor Vergata, Computer Engineering Department as a researcher. He has held computer science research appointments with the (Italian) National Research Council, and visiting scientist appointments with the IBM T.J. Watson Research Center, Yorktown Heights, New York. His research interests include high performance computing, parallel and distributed systems, Web systems and infrastructures, load balancing, performance analysis. In these fields he has published more than 70 papers in international journals, book chapters and conference proceedings. He has served as a member of organizing or program committees of national and international conferences (recently, World Wide Web and Sigmetrics).

Salvatore Tucci Salvatore Tucci received the Laurea degree (master) in physics from the University of Pisa, Italy in 1972. From 1973 to 1975 he held a post-graduate fellowship in computer science at IEI-CNR, Pisa. In 1975, he joined the Department of Computer Science at University of Pisa as Assistant Professor. In 1987 he joined

the Department of Computer Engineering of the University of Roma Tor Vergata as Full Professor. He held visiting position in 1987 at INRIA, Paris, and in 1981/1982 at the IBM T.J. Watson Research Center, Yorktown Heights, NY. From 1990 to 1999 he was Dean of Computer Science curricula at the Faculty of Engineering. His research interests include performance evaluation, parallel and distributed systems, multimedia applications, conception and design of large information systems for decision-making and government. In these fields he has authored more than 60 research papers in referred journals and international conferences. He is currently on leave as the Director of the Office for Informatics, Telecommunications and Statistics of the Italian Prime Minister.