Enhancing a Web-server Cluster with Quality of Service Mechanisms

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

Michele Colajanni, Marco Mambelli
University of Modena
Modena, Italy 41100
{colajanni, mmambelli}@unimo.it

Abstract

*The Web is now a mature and business-oriented media so the need for differentiated classes of users and services is becoming stronger than ever. This differentiation is desired to accommodate heterogeneous application requirements and user expectations, and to permit differentiated pricing for content hosting or service providing. In this paper, we introduce the concept of "Quality of Web-based Services" (*QoWS*) which is inspired by the well known QoS principles for networks, but it is focused on the server side of the Web system. In particular, we investigate how it is possible to enhance with QoWS mechanisms a Web site that is hosted on a system platform consisting of locally distributed server nodes, namely Web-server cluster or* Web cluster *in short. We discuss and compare some QoWS policies and mechanisms that can be used to transform a best-effort Web cluster into a QoWS-enhanced system. We determine a set of confident Service Level Agreements (SLAs) in a Web cluster for different classes of users and Web services. Moreover, we verify through a large set of simulation experiments under realistic workload models which mechanisms are valid for satisfying the pre-determined SLA target.*

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

1 Introduction

A significant amount of research on Quality of Service (QoS) has focused on the network infrastructure. However, network QoS alone is not sufficient to support end-to-end QoS. To avoid the situation where high priority traffic reaching a server is dropped at the server side, the system hosting the Web site should be also enhanced with mechanisms for delivering end-to-end QoS to some classes of users and services. When many users rely on the Web for up-to-date information and business processes, it is necessary to pass from a best-effort system to a system with guaranteed performance. In this paper, we introduce the concept of "Quality of Web-based Services" (*QoWS*) which is inspired by the well known QoS principles for networks (service classification, performance isolation, high resource utilization, and request admission), but it is focused on the server components of the Web. In particular, we investigate how QoWS mechanisms and principles can be applied to a Web site that is hosted on a system platform consisting of locally distributed Web server nodes, namely Web-server cluster or *Web cluster* in short. Given a Web cluster that provides static and dynamic content, we first propose some QoWS policies and mechanisms that can be easily implemented in the considered Web cluster. We then propose a way to setup and tune a set of confident Service Level Agreements (SLAs) for two classes of users and two main classes of Web services. We demonstrate that in this QoWS-enhanced Web cluster the performance for both static and dynamic services conform to the pre-determined SLA.

QoWS-enhanced systems for Web services have been recently proposed for single and multiple server platforms. For a Web site hosted on a single server node, the main mechanisms focus on scheduling algorithms and resource management of server components, either at the HTTP server or kernel level [3, 4, 8, 12]. The proposals for enabling QoWS in cluster-based systems can refer to a single Web site or to multiple Web sites co-located on the same platform (namely, *Web content hosting*). Most results for QoWS-enhanced clusters address the latter issue. On the other hand, in this paper we investigate QoWS mechanisms and policies for a cluster that hosts one popular Web site. One of the first study on this subject is by Chen et al. [7]. Their simulation results clearly show that when the system is highly utilized, differentiated services provide better performance than those achieved by traditional Web clusters. Kanodia et al. have proposed a QoWS policy that uses both admission control and performance isolation mechanisms to guarantee that different classes of service have latencies within pre-specified targets [9]. Unlike our paper that focus on static and dynamic requests, they consider Web sites providing static content only. Two dynamic resource partitioning algorithms for static and dynamic Web requests

have been proposed in [5, 15]. Their experiments demonstrate that dynamic server partitioning always outperforms static server partitioning. An interesting work from Aron et al. has extended the resource principal abstraction to multi-node Web servers [2]. Several companies commercialize as their most recent products content-aware Web switches which can be used for service differentiation in Web clusters (e.g., Nortel Networks' Alteon WebOS, F5's BIG-IP, Resonate's Central Dispatch). These switches provide only very simple service differentiation mechanisms which aim to statically partition server nodes and assign different classes of requests to different server subsets. Various results in literature [5, 15] demonstrate that static partitioning policies cannot adapt to fluctuating arrival rates and servers load conditions. Moreover, it may lead to waste of resources when some partitions are not fully utilized while others might be overloaded.

The rest of this paper is organized as follows. Section 2 describes the components and main dispatching mechanisms in the considered Web cluster architecture. Section 3 discusses some policies and mechanisms that enhance the Web cluster with QoWS principles. Section 4 describes how to choose the right parameters to enforce SLA targets in a Web cluster. The simulation experiments described in Section 5 aim to verify how the proposed policies for QoWS behave when different Web cluster components are subject to stress testing. Finally, Section 6 presents some concluding remarks.

2 Web cluster architecture

A Web cluster refers to a Web site publicized with one name (e.g., www.foo.com) that uses two or more server machines housed together in a single location to handle user requests. A modern Web cluster has typically a front-end component, which we call *Web switch*, acting as a network representative for the Web site. In literature, this component is denoted through various definitions. Figure 1 shows the architecture of a typical Web cluster, where the Web switch implements also some access control on requests.

A Web cluster provides to the external world a single virtual IP address (*VIP*) corresponding to the IP address of the Web switch. The authoritative DNS server(s) for the Web site always translates the site name into the IP address of the Web switch, which receives from clients all inbound packets destined to the VIP address. The Web switch includes a dispatching algorithm to select the Web server node best suited to respond and a dispatching mechanism to route the client request to the target node. The Web switch can operate request assignment at *layer-4* or *layer-7* of the OSI stack [14]. The basic idea is that a layer-4 Web switch is *content information blind*, because it determines the target Web server when the client establishes the TCP connection,

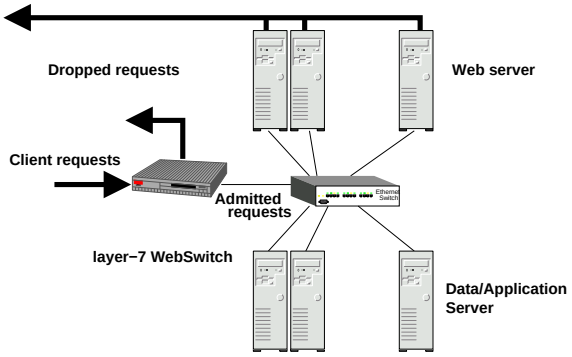

Figure 1. Web cluster architecture with a layer-7 Web switch.

before sending out the HTTP request. On the other hand, a layer-7 Web switch is *content information aware* because it first establishes a complete TCP connection with the client and then examines the HTTP request content (URL) before taking any dispatching or QoWS decision.

In this paper, we consider the Weighted Round Robin (WRR) and the Client Aware Policy (CAP) [6] that are representative examples of dispatching algorithms working at layer-4 and layer-7, respectively. Layer-7 policies have several potential advantages [13], but content aware routing introduces an additional processing overhead at the dispatching entity and may cause the Web switch to become the system bottleneck, thus limiting cluster scalability.

To limit Web switch operations, we consider a *one-way* architecture with a layer-7 Web switch as it puts less overhead on the Web switch, thus guaranteeing higher scalability and throughput than *two-way* architectures. In *one-way* architectures, requests reach the Web switch, but responses flow directly from the servers through another Internet connection. Our Web cluster consists of multiple Web servers and back-end servers, and a dedicated machine that acts as a Web switch. All them are connected through a switched Fast Ethernet, as in Figure 1. We use real parameters to setup the Web cluster components. For example, the disk is parameterized with the values of a 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. The network interface is a 100Mbps Ethernet card. The Web server software is an Apache 1.3 server, where an HTTP daemon waits for connection requests. Each client, after activation, enters the system and generates the first connection request to the Web switch of the cluster. The entire period of connection to the site, namely *Web session*, consists of one or more requests. At each request, the Web switch applies some dispatching pol-

icy (enhanced with some QoWS mechanisms) to determine a target Web server. Some objects may be *dynamic* that is, they require some computation and/or database search on the back-end servers. Table 1 summarizes the main characteristics of the Web cluster.

Parameter	*Value*
Number of Web servers	10
Number of back-end servers	10
Disk transfer rate	20 MBps
Memory transfer rate	100 MBps
Intra-servers bandwidth	100 Mbps
Server state gathering interval	10 sec.
HTTP protocol	1.1

Table 1. Parameters of the Web cluster.

3 Enhancing the Web cluster with QoWS mechanisms

The traditional goal of dispatching policies and mechanisms used in Web clusters lies in providing best-effort services through load sharing. Neither service differentiation nor QoS mechanisms are typically implemented. In this paper, we integrate the load sharing functionality of the Web switch with some QoWS mechanisms. The motivation is that the front-end switch of a Web cluster has a centralized control on the system status and can provide fine-grained control on request assignment. At the base of QoS (and QoWS as well) there are the concepts of *service* and *Service Level Agreement* (SLA). A *service* defines a set of characteristics significant for the Web content provision, specified in quantitative or statistical terms. We refer to *service class* to denote the differentiation of incoming requests and users into classes. The proposed policies support multiple service classes. To make the presentation lighter, in this paper we consider without loss of generality two classes of users denoted as *high* and *low* classes, and two main classes of requests that is, *static* and *dynamic*.

The need of QoS was born in the computer network area, where this topic has been widely investigated. In [11] the authors formulate the basic principles necessary to provide QoS guarantees to network applications: *service classification*, *performance isolation*, *high resource utilization*, and *request admission*. This last function includes a declaration step, a verification of the availability of resources for satisfying the request requirements, and an access control step.

Admission control has been proposed as a first key mechanism to prevent performance degradation of Web services [12, 8]. Both admission control and scheduling can be based on the concept of resource principal that is, the maximum amount of resources assigned to a service class. In a single-node Web server, Banga et al. [3] have studied how to ensure performance isolation by using resource containers which provide an abstraction from resource principals. In [5] we found that the best way to guarantee all basic QoS principles for QoWS in a Web cluster is an approach that uses a layer-7 Web switch to implement request classification and admission control mechanisms, and a *dynamic server partition* algorithm for performance isolation and high utilization mechanisms.

Dynamic server partition denotes a large class of QoWS policies. In the hypothesis of two classes of users, we partition the servers into two sets, denoted as *High Set* and *Low Set*. Incoming requests classified as high are assigned to servers in the first set, while servers in the Low Set must serve user requests belonging to the low class. Besides service classification and admission control, the policies for dynamic server partition include a mechanism to dynamically adapt the sizes of the two sets to the actual workload composition and servers' load state. In this paper, we describe how it is possible to implement one policy of this class, namely **DynamicPart**, and some variants of it. In the basic version, the admission control policy can reject requests belonging to the low class only. Once the request has been admitted into the system, the Web switch uses the dynamic WRR policy to select the target server in the corresponding set. For a more accurate description, let us consider a Web cluster with N servers, and denote with $HS(t)$ and $LS(t)$ the cardinality of the High Set and Low Set at time t, respectively. $HS(t)$ is initially set to $HS(0)$, then the QoWS algorithm adapts it dynamically to the load conditions. The issue is to determine the best value for $HS(t)$ that allows the Web cluster to satisfy the SLA contractual targets for high class users.

$HS(t)$ is a function of several parameters, such as the (Web and back-end) server capacity, the Y value for the SLA, and the workload. In the next section we demonstrate that the main impact on Web cluster performance is by far due to dynamic requests. Hence, the focus of the DynamicPart algorithm(s)[1] will be on the maximum number of connections containing dynamic requests (namely, $MaxDynConn$) that the system can sustain with a latency time parameter less or equal to Y seconds.

An important observation is in order. Since each dynamic request is processed by only one Web server and one back-end server, the $MaxDynConn$ parameter is independent of the number of servers in the Web cluster. Hence, DynamicPart chooses $HS(t)$ such that each server belonging to the High Set has to serve a number of dynamic requests less or equal to $MaxDynConn$.

Assuming that the distributions of the service time for static and dynamic requests are known (benchmark tests for the Web services can be easily carried out by the ser-

[1] Unlike the DynamicPart algorithm proposed in [5], the version proposed in this paper considers as the load metric only the dynamic requests.

vice provider), four parameters remain at time t: the total number of client requests $TotalConn(t)$, the percentage of dynamic requests $\delta(t)$, and the percentage of high class requests $\rho(t)$, and the SLA value. The number of servers in the High Class at time t results from the following equation:

$$HS(t) = \left\lceil \frac{\rho(t) \cdot \delta(t) \cdot TotalConn(t)}{MaxDynConn} \right\rceil \qquad (1)$$

It is evident that if $HS(t) > N$, the Web cluster is underprovisioned. We also exclude the case of $HS(t) = N$ because at least one server should be in the Low Set.

DynamicPart sets $HS(0)$ on the basis of the expected percentage of dynamic requests from the high class users. Then it dynamically adjusts it to let the system satisfy the SLA under the load at time t. (In our experiments, $HS(t)$ is evaluated every 10 seconds.) If $HS(t) > HS(t-1)$ DynamicPart adds to the High Set the least loaded server(s) of the Low Set. From that point, the moved server(s) will receive only new requests from the high class users, and will continue to serve requests of already accepted connections belonging to the low class. (Stronger actions can be used, such as dropping all pending low class requests, but we did not use them because highly unfair solutions.) If $HS(t) < HS(t-1)$, the server(s) with the lowest number(s) of high user requests will return to the Low Set.

The cardinality of the Low Set of servers is simply given by $LS(t) = N - HS(t)$. However, it is important not to overload these servers, because they must be ready to pass to serve high class users, once they are moved to the High Set. For this reason, we choose a maximum number of connections that each server in the Low Set can accept, namely $MaxConn_{LS}$, and integrate the Web switch with an admission control policy. The Web switch gathers from each server in the Low Set the number of active connections at time t, thus computing the aggregate number $TotalConn_{LS}(t)$, and accepting low class requests only if $MaxConn_{LS} \cdot LS(t) > TotalConn_{LS}(t)$.

In this paper, we consider also two variations of the basic DynamicPart policy: DynamicPart-DRdrop aims to reduce the percentage of dropped requests; DynamicPart-HUdrop allows the system to reject high class requests, if the Web cluster is really overloaded and the SLA requirement for high class users cannot be satisfied.

DynamicPart-DRdrop. The basic idea is to use a request admission control which is based on both the user class and the type of service being requested. As dynamic requests may require service times of two orders of magnitude higher than static requests, dropping few of them can have a beneficial effect on cluster load and prevent the need to deny service to static requests from low class users. When $MaxConn_{LS} \cdot LS(t) < TotalConn_{LS}(t)$, DynamicPart-DRdrop starts to reject the dynamic requests only. If load measurements at time t denote that the low class servers are critically loaded (e.g., $2MaxConn_{LS} \cdot LS(t) < TotalConn_{LS}(t)$), the Web switch rejects static requests too.

DynamicPart-HUdrop. A modified version of DynamicPart-DRdrop is the DynamicPart-HUdrop policy that allows the system to reject high class requests if the SLA requirement risks not to be satisfied. This could occur when the High Set has reached its allowed maximum size and the QoWS policy cannot intervene anymore on dynamic adjustment of the server sets. Dropping high class requests is well motivated in reality because, if the SLA requirement is not respected, to drop a high class request has the same penalty impact on the Web service provider as to fail the SLA target.

4 SLA evaluation for a Web cluster

An SLA is a specified performance contract on which the user and the Web service provider agree. Since rare violations of the SLAs are allowed without catastrophic consequences, we consider most effective to express the SLA for QoWS in terms of *predictive service*. In particular, we choose the 95-percentile of the *latency time of a client request* [10] at the Web cluster as the main metric for SLA. This metric measures the completion time of a request at the Web cluster side and does not include network delays.

In this paper, we consider two treatments for the high class and low class classes of users both issuing static and dynamic requests to the Web cluster. In particular, the SLA for the high class states that the "*95-percentile of the latency time of requests from high class users (in short, high class requests) must be less than a threshold of Y seconds*" while "*low class users continue to receive best effort service*". To complete the specification of the SLA statement, we need to define for the considered Web cluster and expected workload a realistic value for the upper bound on the 95-percentile of the latency time that corresponds to the previously defined *Y* parameter. To this purpose, we evaluate the 95-percentile of latency time of static and dynamic requests under normal load conditions.

In our Web cluster, a static request is served by one Web server, while a dynamic request is served by one Web server and one back-end server for the static and dynamic information, respectively. More sophisticated architectures exist where the embedded objects of a request could be served by different nodes, but we do not consider them in this paper. As each request is served by one Web server and (if necessary) one back-end server, it is important to evaluate the capacity of the single server nodes, independently

of the number of available servers in the Web cluster. For this reason, we first evaluate the *capacity* of the testbed system that is, a not QoWS-enabled Web cluster consisting of one Web server and one back-end server under the *expected workload*. The capacity of a Web cluster consisting of multiple servers is simply obtained by using a multiplicative factors for $MaxDynConn$, because we assume that the Web switch is able to balance the load among the servers in each set (there are several dispatching policies that guarantee good results, for example WRR).

The expected workload model we consider in this paper incorporates all recent results on Web load characterization (e.g., [1]). We consider two main classes of services provided by the Web site that is, *static services* composed by requests for HTML pages with some embedded static objects, and *dynamic services*, where some objects belonging to the page are dynamically generated through Web server and back-end server interactions. The workload model for static requests is described in [5]. A dynamic Web service request is composed by a page request with at most two embedded objects that are generated by the back-end servers. Specifically, we consider the impact of intensive database queries (that is, disk bound requests) and dynamic page composition. If not otherwise specified, the basic workload composition consists of 80% of static requests and 20% of dynamic requests. The service time for a static object is proportional to the file sizes. The service time on the back-end server for a dynamic object is modeled according to a hyper-exponential distribution. Specifically, dynamic requests are further classified as high, medium, and low intensive, on the basis of the computational impact they put on the back-end servers.

We stress the system to find its break-point and a suitable working range to evaluate its performance. We then set SLA parameters and perform some sensitivity analysis on the workload. Figure 2 shows the 95-percentile of the latency time as a function of the client arrival rate for a cluster consisting of one switch, one Web server and one back-end server with the characteristics described in Section 2. This figure clearly shows that static requests (even those issued by the high class users) typically do not represent a problem for the SLA. The real challenge is to set the right SLA for the latency time of dynamic requests.

From this figure, we observe that the 95-percentile of latency time of dynamic requests is about 2 seconds when the system is underutilized (that is, less than 5 clients per second), and increases almost linearly up to 4 seconds for a higher offered load. This is clearly the break-point of the Web system, caused by an overloaded back-end server. As this performance figure is representative of the behavior of a system under stress testing, we recommend to choose the time right before the knee of the curve as the upper bound on the 95-percentile of the latency time. In the testbed system, this value corresponds to 4 seconds. Hence, $Y = 4$ is the value for the parameter defined in the SLA for high class users. However, many other considerations different from performance observations contribute to the choice of the SLA in the reality. The tradeoff is clear. Choosing a lower SLA value allows the service provider to offer (and publicize) a better service to a smaller number of high class users. The opposite is true when a higher SLA value is chosen. The service provider has typically a range of values among which he can choose the best SLA for his Web services. The lower bound for this SLA interval is evaluated with the contribution of the system administrator, because it is denoted by the time to serve the most demanding dynamic request. The upper bound depends more on market considerations.

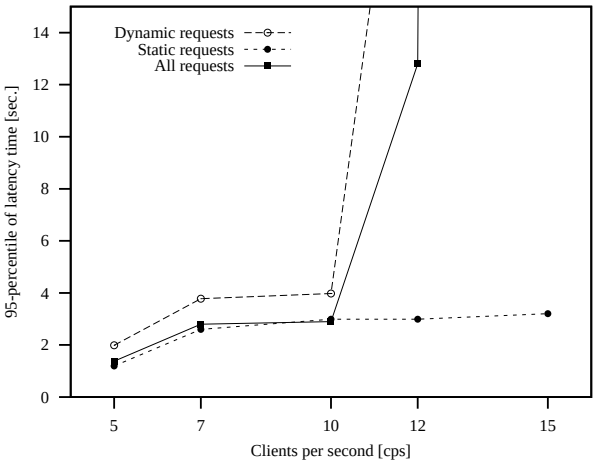

Figure 2. 95-percentile of latency time as a function of the client arrival rate in a single-server system.

The system administrator has to translate the SLA of Y=4 seconds in a metric that the Web switch can use to execute QoWS mechanisms.

This problem exists because in one-way architecture, latency time measurements are not available at the Web switch. Hence, we have to translate the SLA upper bound in practicable metrics to be used in dispatching and admission control policies. From several studies, we can conclude that the throughput expressed as a number of connections for dynamic requests per second is a metric rather representative of the system load. To evaluate the throughput corresponding to the break-point, we measure it as a function of the client arrival rate. From Figure 3 we observe that the arrival rate of 10 clients per second, corresponding in Figure 2 to a latency time of 4 seconds, occurs when the system serves 30 connections per second. (This value corresponds also to the number of page requests per second because we assume to work with the HTTP/1.1 protocol where each TCP connec-

tion carries an entire page request.) A more precise analysis of the throughput evidences that 24 of the 30 requests are for static pages and 6 for dynamic pages. Moreover, Figure 3 shows that the maximum capacity of the Web cluster (reached at 37 requests per second) is clearly due to the dynamic requests because the system would be able to serve many other static requests. Indeed, the throughput for static requests continues to grow linearly. We can conclude that when the number of requests for dynamic pages exceeds the value of 5, the system has a bottleneck on the back-end server. Furthermore, it is quite easy to let the Web switch know the current number of connections. This justifies the choice for $MaxDynConn = 5$ as a break-point indicator for the system composed of one Web server and one back-end server.

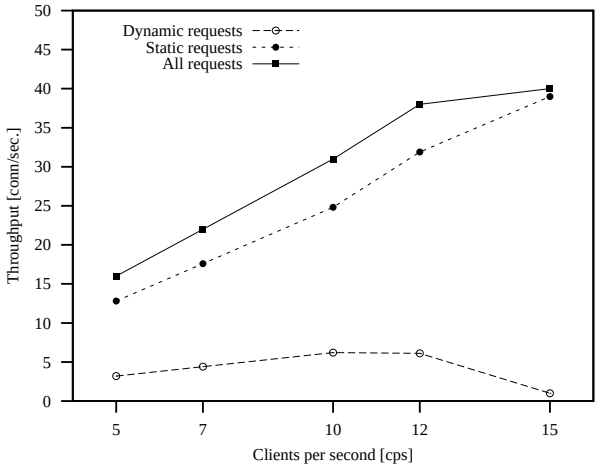

Figure 3. Throughput as a function of the client arrival rate in a single-server system.

The previous stress testing analysis has been based on the number of request arrivals to the system. It is important to observe that there are other two potential stress factors for the Web cluster: the percentage of dynamic requests, and the percentage of high class users. These three parameters denote the admissible space where the Web cluster can guarantee SLA targets. Workload values beyond the bounds of the admissible space cannot be faced by QoWS mechanisms and policies, but they require interventions on the system, e.g., adding some server nodes.

Space limits do not consent us to enter into many details, however to give some ideas, in Figure 4 we report a sensitivity analysis based on the percentage of dynamic requests while keeping the percentage of high class users equal to 20%. This figure shows that an augment of dynamic requests from 5% to 25% causes only a slight increase in the 95-percentile of latency time. This interval denotes the acceptable range for the request mix of the considered workload and Web cluster. Henceforth, the experimental results will refer to a percentage of dynamic requests equal to 20%.

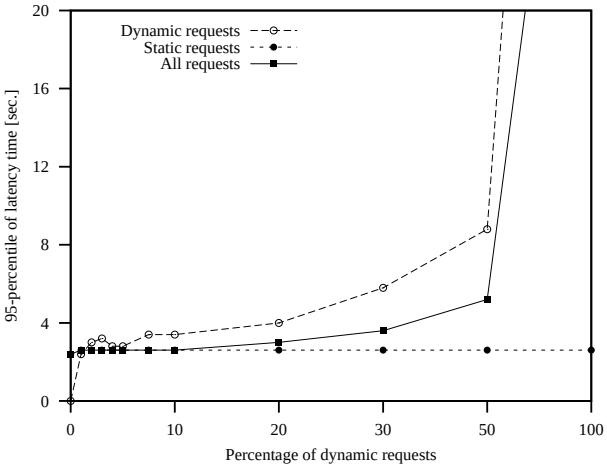

Figure 4. Latency time as a function of the percentage of dynamic requests in a single-server system.

5 Simulation analysis

Unlike a traditional performance analysis where the goal is to evaluate which policy provides best performance results (e.g., minimum response time, maximum throughput), the goal of the QoS analysis is to verify whether the QoWS-enhanced Web cluster satisfies the SLA target for different scenarios. A secondary goal is to compare the behavior of the different proposed QoWS policies. Being the SLA the most important target, a policy that satisfies it for all experiments is preferable to a policy that obtains a lower latency time in most instances but it is unable to guarantee the SLA in others. We compare the proposed QoWS policies under different workload scenarios applied to a Web cluster. By varying the amount and the composition of the offered load, we stress different components of the system. Stress testing analysis is motivated by the preliminary observation that if the Web cluster components are underutilized, most policies can easily satisfy SLA. If not otherwise specified, we consider the Web cluster and the workload model described in Sections 2 and 4.

5.1 Stress testing on back-end server nodes

In this section we compare the performance of the QoWS policies when the back-end nodes of the Web cluster represent the system bottleneck. Figure 5 shows the 95-percentile of latency time of the proposed dynamic partition algorithms and compare them with CAP policy [6] as representative for QoWS-blind algorithms operating at

layer-7. As a first result, we observe that DynamicPart and DynamicPart-Drdrop satisfy the SLA of 4 seconds for high class users, while the same target is not achieved by the CAP policy.

Further, we observe that both QoWS policies are able to guarantee SLAs, but their performance differ in terms of the latency time and the number of dropped requests experimented by low class users. In particular, Figure 5 shows that DynamicPart-DRdrop performs slightly better than DynamicPart in terms of 95-percentile for high class requests. On the other hand, DynamicPart achieves better results for low class requests with a page latency time that ranges from 4.2 to 4.4 seconds, while DynamicPart-DRdrop obtains a latency time for low class requests beyond 5 seconds. The motivation for this result is that DynamicPart-DRdrop rejects only dynamic requests. This entails a larger number of Web servers devoted to the high class partition. As a result, the servers in the low set are overloaded with respect to the case in which the Web switch drops both static and dynamic requests originated by low class users. DynamicPart drops a higher number of low class requests with respect to DynamicPart-DRdrop, which rejects only those low class requests that really impact on the system performance that is, the requests for Web pages containing dynamically generated Web objects. Indeed, DynamicPart drops 7% of low class requests when the systems is overloaded, while DynamicPart-DRdrop drops less than 4% of low class requests (Figure 6).

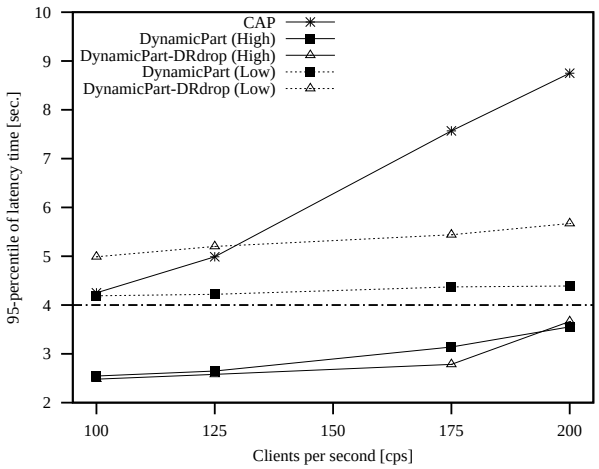

Figure 5. Sensitivity of latency time to the client arrival rate for QoWS-aware and QoWS-blind algorithms.

5.2 Setting stronger SLA requirements

Figure 5 shows that all the presented dynamic algorithms are able to satisfy the SLA. We did not report in Figure 5

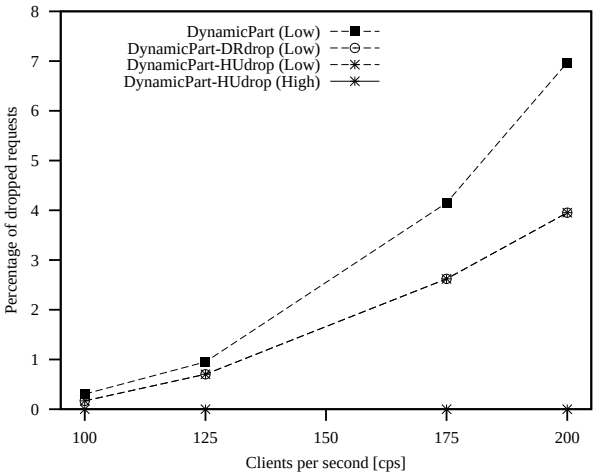

Figure 6. Sensitivity of number of dropped requests to the client arrival rate for QoWS-aware and QoWS-blind algorithms.

the results achieved by DynamicPart-HUdrop as its behavior is analogous to DynamicPart-DRdrop (the two curves overlap). As the SLA for the high class is never violated, DynamicPart-HUdrop does not activate the mechanism to drop high class requests.

Adapting to changing conditions is critical to the success of a QoWS-enabled system. There are many parameters and workload characteristics that can change in the Web. Space limits do not allow to carry out an extensive evaluation of all feasible variations. Hence, in this section we focus on behavior of QoWS policies when the SLA becomes more severe than that evaluated in the previous section.

To test the system with a stricter SLA, we impose for high class requests a new stricter SLA with an upper bound of 2.5 seconds on the 95-percentile of latency time. If we consider that the basic latency time for a dynamic request in a under-utilized system is about 2 seconds, the new SLA requires that a dynamic request has to be served in a margin of 25% on the basic latency time with 0.95 probability. (When the SLA was previously set to 4 seconds, the margin was 100% higher than the basic latency time.)

Figure 7 shows that the DynamicPart and DynamicPart-DRdrop policies are not able to respect the new SLA set to 2.5 seconds. Therefore, we analyze what happens when we consider the DynamicPart-HUdrop policy that may even drop high class requests, if actions on low class requests are not sufficient to guarantee QoWS.

Figure 7 reports the latency time achieved by the DynamicPart-HUdrop policy, in which any Web server can drop high class requests if the SLA requirement for these users cannot be satisfied. This figure shows that the DynamicPart-HUdrop policy achieves a 95-percentile of la-

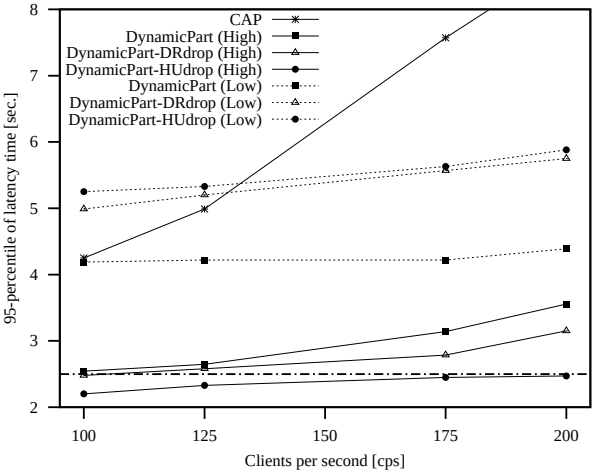

Figure 7. Latency time with SLA requirement stress.

tency time lower than 2.5 seconds, at the price of an acceptable percentage of dropped high class requests. The percentage of dropped high class requests is 4%, while the percentage of dropped requests in the low class is very high. However, this result is acceptable, if one considers that low class requests are served under a best effort policy and that the system is highly overloaded. Indeed, the growth of the latency time for the CAP policy shows that 125 clients per second is the break-point of the system and we are overloading the system up to 200 clients per second.

6 Conclusions

In this paper we analyze mechanisms and policies for enabling Quality of Service principles in cluster-based Web systems. We first propose and compare various policies for dynamic server partitioning that satisfy all QoS principles. We then focus on the parameter to determine a set of confident SLAs and evaluate a Web cluster with different classes of users and Web services. Our simulation experiments show that the proposed class of policies that dynamic adapt system resources devoted to the most demanding class of users is able to meet the Service Level Agreement for different load and system conditions.

Acknowledgments

The authors acknowledge the support of the Ministry of University and Scientific Research in the framework of the project “Sistemi Web ad elevata qualità del servizio”. The first two authors also acknowledge the support of Banca di Roma (Res. contract BDR-2001 on “Advanced technologies”).

References

[1] 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.

[2] 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*, Santa Clara, CA, June 2000.

[3] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. *World Wide Web*, 2(1-2), 1999.

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

[5] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. *ACM Performance Evaluation Review*, 29, 2001.

[6] 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.

[7] X. Chen and P. Mohapatra. Providing differentiated service from an Internet server. In *Proc. IEEE Int’l Conf. on Computer Communications and Networks*, Boston, MA, Oct. 1999.

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

[9] 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.

[10] D. Krishnamurthy and J. Rolia. Predicting the QoS of an electronic commerce server: Those mean percentiles. In *Proc. of Workshop on Internet Server Performance*, Madison, WI, June 1998.

[11] J. F. Kurose and K. W. Ross. *Computer Networking: A top down approach featuring the Internet*. Addison-Wesley Longman, 2000.

[12] K. Li and S. Jamin. A measurement-based admission-controlled Web server. In *Proc. of IEEE Infocom 2000*, Tel Aviv, Israel, Mar. 2000.

[13] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and N. E. 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, Oct. 1998.

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

[15] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *Proc. of IEEE Infocom 2001*, Anchorage, Alaska, Apr. 2001.