A Performance Study of Distributed Architectures for the Quality of Web Services*

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

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

Abstract

The second generation of Web sites provides more complex services than those related to Web publishing. Many users already rely on the Web for up-to-date personal and business information and transactions. This success motivates the need to design and implement Web architectures being able to guarantee the service level agreement that will rule the relationship between users and Web service providers. As many components of the Web infrastructure are beyond the control of Web system administrators, they should augment satisfaction percentage of the assessed service levels by relying on two mechanisms that can be integrated: differentiated classes of services/users, Web systems with multi-node architectures. The focus of this paper is on this latter approach. We review systems where replicated Web services are provided by locally and geographically distributed Web architectures. We consider different categories of Web applications, and evaluate how static, dynamic and secure requests affect performance and quality of service of distributed Web sites.

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. Many users already rely on the Web for up-to-date personal, professional and business information. The substantial changes transforming the World Wide 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. Users are not willing to tolerate latency times greater than eight-ten seconds. Furthermore, their tolerance for latency decreases over the duration of interaction with a site. This new scenario motivates the need to design and implement architectures being able to guarantee the *service level agreement* (SLA) that will rule the relationship between users and Web service providers. The users do not know neither care of complexity of Web infrastructure and technology. They complain if the response time becomes too high, there are many periods of unavailability, the security is not fully guaranteed. Because of the complexity of Web infrastructure, many components could affect the quality of Web services. Hence, the assessed service levels for all SLA parameters would require interventions on each component of the Web: from network technology and protocols, to hardware and software architectures of Web servers and proxies. As most components of the Web infrastructure are beyond the control of Web system administrators, quality of Web services is very hard to achieve. Network carriers that have a full control on their backbones can provide SLAs contracts with their customers based on network availability and guaranteed network response times. Web service providers cannot guarantee analogous contracts because their actions are limited to a small part of the Web infrastructure. We consider solutions for Web service providers that can act only on their Web systems. To augment satisfaction percentage of the assessed service levels, they can rely on two classes of actions that are not mutually exclusive:

Differentiated Web services. It requires the definition of classes of users/services, choice of the number of priority levels, guarantee of different SLAs through priority dispatching disciplines [6, 14, 19] and monitors for starvation of low priority services.

Architecture design. The goal is to find the right architecture that guarantees the SLA on all Web users/services. The three directions are: *scale-up* by adding memory and CPU power to the single server, *local scale-out* by replicating servers in a local area, *global scale-out* by replicating servers in a geographical context.

The focus of this paper is on the architecture design, while we leave to future work the combination of the two previous solutions. As example of applications for stress testing, we consider three categories of Web sites: Web publishing sites with static pages, Web sites with static and dynamic pages, e-commerce sites with some percentage of secure requests that typically have the most severe SLA parameters. When these Web services are implemented on top of locally and geographically distributed Web systems, accurate design and sophisticated algorithms for traffic control, load balancing, and request dispatching are necessary. We analyze performance and scalability of distributed Web sites that have to guarantee the assessed SLAs for different Web services even under high traffic conditions. We discuss efficiency and limitations of proposed solutions and compare how different architectural approaches satisfy SLA performance requirements.

The rest of the paper is organized as follows. In Section 2, we outline the differentiated Web service solution to achieve the assessed SLA for the quality of Web services. In Section 3 and 4, we propose a classification of *locally* and *geographically* distributed Web architectures, respectively. In Section 5 and 6, we describe the system model and workload we use for the simulation analysis. In Section 7, we present and discuss the results of the analysis for three classes of Web sites with a mix of static, dynamic and secure requests. In Section 8, we outline our conclusions and future work.

2. Differentiated Web services solutions

Most proposals for guaranteeing quality of Web services look at new Web server architectures that can support differentiated scheduling services to enable preferential treatment of classes of users and services. The main motivation is that first-come-first-served service policies implemented by traditional Web servers can undermine any improvements made by network differentiated service [6]. Since overloaded servers affect all requests in the same manner, a FCFS discipline makes impossible to guarantee SLAs to preferred clients. To overcome this drawback, priority-based scheduling schemes can be implemented in the Web server to provide differentiated SLAs.

The main components of a Web server architecture that provide differentiated service must include a *classification* mechanism to assign different priority classes to incoming requests, an *admission control* policy to decide how and when to reject requests according to their priorities, a *request dispatching* policy that decides the order in which requests should be serviced, and a *resource dispatching* policy to assign server resources to different classes of priority [6]. Most proposed architectures modify Web servers at application or kernel level to allow differentiated control through dispatching of requests and resources.

A commercial system such as HP's WebQos [6] provides quality of service by using priority levels to determine admission priority and performance level. The method used to dynamically classify requests on a per-session basis includes source IP address, TCP port number, and the requested content. Similar Web server prototypes that support differentiated services have been proposed in [10, 19].

To enforce SLA constraints, Pandey et al. [14] examine selective allocation of server resources through the assignment of different priorities to page requests. Menasce et al. [12] analyze and compare policies that dynamically assign priorities to customers of a commercial Web site by differentiating between visitors and potential buyers.

Most of the previous results consider Web sites consisting of a single server node. On the other hand, we claim that popular Web sites cannot rely on a single powerful server to support SLA for ever increasing request load. Scalability, load balancing, and dependability can be only provided by multiple Web server architectures that distribute intelligently client requests across multiple server nodes. The main components of a typical multi-node Web system include a *dispatching mechanism* to route the client request to the target Web server node, a *dispatching algorithm* to select the Web server node best suited to respond, and an *executor* to carry out the dispatching algorithms and support the relative mechanism. The decision on client request assignment can be taken at various network levels. In the following sections, we propose a classification of existing approaches based on the type of distribution of the server nodes that compose the scalable architecture that is, *local* distribution and *global* distribution. We limit our attention on Web sites that use a single URL to make the distributed nature of the service transparent to the users.

3. Locally distributed Web systems

A locally distributed Web server system, namely *Web cluster*, is composed by a tightly coupled architecture placed at a single location. The Web cluster is publicized with one URL and one virtual IP address (*VIP*). This is the IP address of a *Web switch* that acts as a centralized dispatcher with full control on client requests. The switch receives the totality of inbound packets for the VIP address and distributes them among the Web servers through the mapping from VIP to the actual server address. The goal is to share the load, and avoid overloaded or malfunctioning servers. The Web switch is able to identify univocally each Web server through a private address, that may correspond to an IP address or to a lower-layer (MAC) address.

Web clusters can provide fine grain control on request assignment, high availability and good scalability. Implementations can be based on special-purpose hardware devices plugged into the network or on software modules running on a common operating system. The architecture alternatives can be broadly classified according to the OSI protocol stack layer at which the Web switch operates the request assignment, that is, *layer-4* and *layer-7* Web switches. The main difference is the kind of information available to the Web switch to perform assignment and routing decision.

- Layer-4 Web switches are *content information blind*, because they determine the target server when the client establishes the TCP/IP connection, before sending out the HTTP request. 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, SYN/FIN flags in the TCP header.

- Layer-7 Web switches can deploy *content information aware* distribution, by letting the switch establish a complete TCP connection with the client, examine the HTTP request and then relay the latter to the target server. The selection mechanism can be based on the Web service/content requested, as URL content, SSL identifiers, and cookies.

Layer-4 Web switches work at TCP/IP level. 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 a new or an existing connection. Layer-4 Web switches can be classified on the basis of the mechanism used by the Web switch to route inbound packets to the target server and the packet way between the server and client. The main difference is in the return way that is, server-to-client.

In *two-ways* architectures both inbound and outbound packets are rewritten at TCP/IP level by the Web switch. Packet rewriting is based on the IP Network Address Translation approach: the Web switch modifies inbound packets by changing the VIP address to the IP address of the target server, while it rewrites the server IP address with the VIP address in outbound packets. Furthermore, the Web switch has to recalculate the IP and TCP header checksum for both packet flows. In *one-way* architectures only inbound packets flow through the Web switch, thus allowing a separate high-bandwidth network connection for outbound packets. The routing to the target server can be accomplished by rewriting the IP destination address and recalculating the TCP/IP checksum of the inbound packet or by forwarding the packet at MAC level [11].

Layer-7 Web switches work at application level, thus allowing content-based request distribution. The Web switch must establish a TCP connection with the client and inspect the HTTP request content prior to decide about dispatching. The potential advantages of layer-7 Web switches include increased performance due to higher cache hit rates [13, 18], the ability to employ specialized Web server nodes and partition the Web content among the servers [20]. However, content aware routing introduce an additional processing overhead at the dispatching entity and may cause the Web switch to become the system bottleneck, thus limiting cluster scalability [3, 18]. Similarly to the layer-4 solutions, layer-7 Web switch architectures can be classified on the basis of the mechanism used by the switch to redirect inbound packets to the target server and the way back of packets from server to client.

In *two-ways* architectures, outbound traffic must pass back through the switch. The proposed approaches differ in the way requests are routed from the Web switch to the target server. In the *TCP gateway* approach, an application level proxy located at the switch mediates the communication between client and server; the *TCP splicing* approach is an optimization of TCP gateway in that data forwarding occurs at network level. In *one-way* architectures the server nodes return outbound traffic to the client, without passing through the Web switch. This is achieved by allowing the Web switch to hand-off the TCP connection to the selected server [13].

4. Globally distributed Web systems

Upgrading content site infrastructure from a single node to a locally distributed system provides a limited relief because the network link of the Web site to Internet may become the bottleneck. In order to reduce network impact on users' response time and to scale to large traffic volumes, a better solution is to distribute Web servers over the Internet, namely *global scale-out*. In this section we consider two classes of globally distributed Web systems: *distributed Web servers* and *distributed Web clusters*.

A distributed Web servers system consists of geographically distributed nodes, each composed of a single server. In these architectures the requests assignment process can occur in two steps: a first dispatching level where the authoritative Domain Name Server (DNS) of the Web site or another centralized entity selects the target Web server, and a second dispatching level carried out by each Web server through some request redirection mechanism.

DNS-based dispatching was originally conceived for locally distributed Web systems. It works by intervening on the address lookup phase of the client request. Load sharing is implemented by translating the site hostname into the IP address of the selected Web server. When the authoritative DNS server provides the address mapping, it can use various dispatching policies to select the best server, ranging from simple static round-robin to more sophisticated algorithms that take into account both client and server state information [7]. Most implemented distributed Web servers evaluate client-to-server network proximity, so that the DNS can return the IP address of the server closest to the user [9]. The goal is to limit the network latency component in the response time.

The main problem of DNS dispatching is its limited control on workload reaching the Web site, because of hostname-to-IP caching occurring at various network levels. In particular, the authoritative DNS of highly popular sites can provide only a very coarse distribution of the load among the Web servers, as it controls less than 5-7% of requests reaching the Web site. Furthermore, heterogenous Web traffic arrivals due to domain popularity and world time zones are highly amplified by the geographical contest. Indeed, a geographically distributed Web site that tends to serve closest requests only, may risk to be highly unbalanced because the amount of request from an Internet region is strictly dependent on day time. The consequence of time zones and proximity algorithms alone is to have one or two highly loaded servers in two regions and other almost idle servers. To address DNS (centralized) dispatching issues we can add a second level dispatching mechanism. The most common is a distributed dispatching policy that is carried out by the critically loaded Web servers through some redirection mechanisms, for example HTTP redirection [7], or IP tunneling [5].

As an alternative solution we consider a *distributed Web clusters* system consisting of geographically distributed nodes, each composed of a cluster of servers. A distributed Web cluster has one hostname and an IP address for each Web cluster. We suppose that requests to a Web cluster are scheduled through one of the mechanisms described in Section 3. Here, we focus on request management among the Web clusters. We can distinguish the proposed architectures on the basis of dispatching levels, typically two or three. The first level dispatching among the Web clusters is typically carried out by the authoritative DNS of the Web site or another entity that implements some proximity dispatching strategy The second level dispatching is carried out by the Web switches that dispatch client requests reaching the cluster among the local Web server nodes. Most commercial products that provide global load balancing implement this class of architectures [9, 11, 16].

The main problem is that dispatching algorithms based on network proximity are not able to react immediately to heavy load fluctuations of Web workload that are amplified by the geographical context. Therefore, it seems convenient to integrate the two level dispatching architecture with a third level assignment activated by each Web server through the HTTP redirection mechanism [8]. This third level dispatching mechanism allows an overloaded Web cluster to easily shift away some portion of load assigned by the first dispatching level. The third dispatching level is necessary to guarantee scalability and load balancing of geographically distributed Web sites, and to enhance quality of Web services by augmenting the percentage of requests with guaranteed response time. On the other hand, request redirection should be used selectively because additional round-trip time risks to increase latency time experienced by users. We investigate some mechanisms to limit request reassignments in [8].

5. System models

The main goal of our analysis is to find the main characteristics of the architecture that can guarantee the SLA on **all** Web services. To this purpose, we investigate two directions for system design that is, *local scale-out* by replicating Web servers in a local area, and *global scale-out* by replicating Web servers in a geographical context. We consider a selection of the previously described multi-node architectures. In particular, we focus on Web clusters with layer-4 Web switch (for sites with static and dynamic requests) and layer-7 Web switch (for sites with secure requests) for local scale-out, and distributed Web clusters for global scale-out.

5.1 Web cluster

A Web cluster system consists of a front-end that acts as a (layer-4/layer-7) Web switch and two levels of server nodes. The nodes in the first tier work as Web servers, while the back-end servers on the second level work as application or database servers (Figure 1). The authoritative DNS server translates the hostname site into the IP address of the Web switch. The addresses of internal server nodes are private and invisible to the extern. The Web switch, Web servers, and back-end servers are interconnected through a local fast Ethernet with 100 Mbps bandwidth. The data flow in Figure 1 shows that the Web switch assigns client requests to a Web server node that cooperate with back-end nodes to produce responses to dynamic requests. We suppose that each Web server stores the same document tree and that each back-end node provides the same services. As the focus is on Web cluster performance, we did not model the details of the external network. To prevent the bridge to the external network from becoming a potential bottleneck for the Web cluster throughput, we assume that the system is connected to the Internet through one or more large bandwidth links that differ from that of the Web switch [11].

Each Web server in the cluster is modeled as a separate CSIM process [17]. Each server has its CPU, central memory, hard disk and network interface. About 15-20 percent of the main memory space of each server is used for Web caching. All above components are resources having their

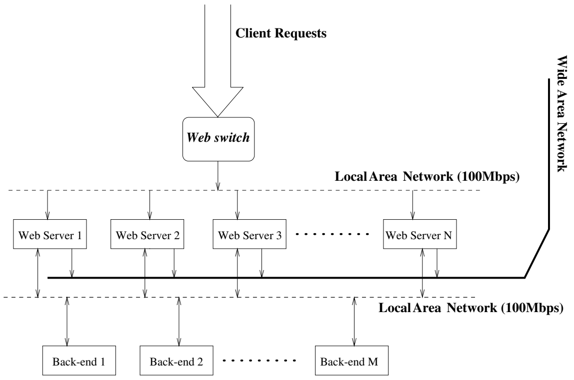

Figure 1. Web cluster architecture.

own queuing systems that allow for requests to wait if CPU, disk or network are busy. We use real parameters to setup the system. For example, the disk is denoted with the values of a real fast disk (IBM Deskstar34GXP) having transfer rate equal to 20 MBps, controller delay to 0.05 msec., seek time to 9 msec., and RPM to 7200. The main memory transfer rate is set to 100MBps. Internal network interface is a 100Mbps Ethernet card. Each back-end server is modeled as a black-box that provides three classes of service with different service time. The parameters for each class are defined in Section 6. The Web server software is modeled as an Apache-like server, that can support secure connection based on Netscape's Secure Socket Layer (SSL). An HTTP daemon waits for requests of client connections on standard HTTP port 80 and on port 443 for secure connections.

5.2 Client - Web site interactions

The interactions of the client with the Web site are modeled at the details of TCP connections including both data and control packets. When a secure connection is requested, we model all details of communication and Web server overhead, due to key material exchange, server authentication, encryption and decryption of public-key and user data. Since HTTP/1.1 protocol allows persistent connections and pipelining, all files belonging to the same Web page request are served on the same TCP connection.

A Web client is modeled as a process that, after activation, enters the system and generates the first TCP connection request to the cluster Web switch. The period of visit of each client to the Web site, namely *Web session*, consists of one or more Web page requests. Each page request is for a single HTML page that may contain a number of embedded objects and may include some computation or database search. The Web switch assigns a new TCP connection request to the target server using the weighted round-robin algorithm [11], while packets belonging to existing connections are routed according to the binding table maintained for each connection. The granularity of dispatching at the Web switch in the static and dynamic scenario is at the client page request level because of the HTTP/1.1 protocol, while it is at the client session level when secure requests are submitted, so as to minimize the authentication overhead required by the SSL protocol. If no HTTP process/thread is available at the Web server, the server forks the HTTP daemon and dedicates a new process for that connection. The client will submit a new page request only after it has received the complete answer that is, the HTML page and all embedded objects. Between two page requests we introduce a user think time that models the time to analyze the requested page and decide (if necessary) for a new request.

The disconnection process is initiated by the client when the last connection of the Web session is closed. The HTTP process of the server receives the disconnection request, closes the TCP/IP connection and then kills itself. The client leaves the system and its process terminates.

5.3 Distributed Web cluster

The distributed Web cluster for global scale-out initiatives consists of an authoritative DNS server and some Web clusters placed in strategic Internet regions. Each cluster is modeled as described in Section 5.1. The DNS server executes the first-level assignment by mapping the hostname into the virtual IP address of one of the Web switches. To reply to the name resolution request issued by the client, the DNS uses a proximity algorithm that assigns the Web cluster closest to the client. The requests arrive then to the Web switch of the target cluster, that executes the second level assignment. We divide the Internet into 4 geographical regions located in different world areas. Each region contains a Web cluster and various client domains. The details of this system are in [8].

6. Workload model

The analysis considers three main classes of load. A Web site may provide one or a mix combination of the following Web services.

Static Web services. Requests for HTML pages with some embedded objects. Typically, this load has a low impact on Web server components. Only requests for very large files are disk and network bound.

Dynamic Web services. Requests for HTML pages, where objects are dynamically generated through Web and back-end server interactions. Typically, these requests are CPU and/or disk bound.

Secure Web services. Requests for a dynamic page over a secure connection. Typically, these services are CPU bound because of overheads to setup a secure connection and to execute cryptography algorithms.

Special attention has been devoted to the workload model that incorporates all most recent results on the characteristics of real Web workload. The high variability and self-similar nature of Web access load is modeled through heavy tail distributions such as Pareto, lognormal and Weibull functions [2, 4, 15]. Random variables generated by these distributions can assume extremely large values with non-negligible probability.

The number of consecutive Web pages a user requests from the Web site (*page requests* per session) follows the inverse Gaussian distribution [15]. The *user think time* is modeled through a Pareto distribution [4, 15]. The number of *embedded objects* per page request including the base HTML page is also obtained from a Pareto distribution [15]. Web files typically show extremely high variability in size. The function that models the distribution of the *object size* requested to the Web site varies according to the object type. For HTML objects, the size is obtained from a hybrid function, where the body follows a lognormal distribution, while the tail is given by a heavy-tailed Pareto distribution [2, 4, 15]. The size distribution of embedded objects is obtained from the lognormal distribution [4]. Table 1 summarizes the parameters' value we use in the so called *static* workload model.

Category	*Distribution*	*Parameters*
Pages per session	Inverse Gaussian	$\mu = 3.86, \lambda = 9.46$
User think time	Pareto	$\alpha = 1.4, k = 1$
Objects per page	Pareto	$\alpha = 1.245, k = 2$
HTML object size	Lognormal Pareto	$\mu = 7.630, \sigma = 1.001$ $\alpha = 1, k = 10240$
Embedded object size	Lognormal	$\mu = 8.215, \sigma = 1.46$

Table 1. Static workload model.

A dynamic request includes all overheads of a static request and overheads due to back-end server computation to generate the dynamic objects. We consider three classes of requests to the back-end nodes that have different service times and occurrence probability. *Light*, *middle-intensive* and *intensive* requests are characterized by an exponential service time on back-end nodes with mean equal to 16, 46 and 150 msec, respectively. The three classes represent 10%, 85%, and 5% of all dynamic requests, respectively. These last parameters are extrapolated by the logfile traces of two real e-commerce sites. Table 2 summarizes the parameters of the so called *dynamic* workload model.

Secure transactions between clients and Web servers involve the SSL protocol. Our model includes main CPU and transmission overheads due to SSL interactions, such as key material negotiation, server authentication, and encryption and decryption of key material and Web information. The

Category	*Mean Service Time*	*Frequency*
Light Intensive	16 msec.	0.1
Medium Intensive	46 msec.	0.85
Intensive	150 msec.	0.05

Table 2. Dynamic workload model.

CPU service time consists of encryption of server secret key with a public key encryption algorithm such as RSA, computation of Message Authentication Code through a hash function such as MD5 or SHA, and data encryption through a symmetric key algorithm, such as DES or Triple-DES. Most CPU overhead is caused by data encryption (for large size files), and public key encryption algorithm (RSA algorithm), that is required at least once for each client session, when the client has to authenticate the server. The transmission overhead is due to the server certificate (2048 bytes) sent by the server to the client, the server hello and close message (73 bytes), and the SSL record header (about 29 bytes per record). Table 3 summarizes the throughput of the encryption algorithm used in the *secure* workload model.

Category	*Throughput (Kbps)*
RSA(256 bit)	38.5
Triple DES	46886
MD5	331034

Table 3. Secure workload model.

The workload models are mixed together to emulate three scenarios: *static scenario* characterized by static workload only; *dynamic scenario* characterized by a mix of static (50%) and dynamic (50%) workload; *secure scenario* characterized by a mix of static (50%) and secure (50%) workload. The secure workload consists of dynamic requests only.

7. Performance analysis

SLA in terms of performance is typically measured as the *K-percentile* of the *page delay* that must be less than Y seconds. Typical measures are 90- or 95-percentile of the requests that must have a delay at the server less than 2-4 seconds, while 7-8 seconds of response time (including also the time for the address lookup phase and network transmission delay) are considered acceptable SLAs at the client side.

In the design of a Web site it is necessary to know the maximum number of clients per second that the system could serve with the requested SLA. We referee to this value as the *break-point* for the Web site. To analyze when the network connection of the Web site to Internet starts to become a bottleneck, we use the *peak throughput* that is, the maximum Web system throughput measured in MBytes per second (MBps). Over certain peaks, it is necessary to pass from a locally to a geographically distributed Web system. However, it is reasonable to consider a geographical distribution even when the throughput in bytes begins to require more than half of a T3 connection that is, 45 Mbps.

In the following sections we discuss a methodology to tune the system for Web system configurations, so as to achieve acceptable SLA performance for the workload scenarios described in Section 6.

7.1 Static scenario

Figure 2 compares the 90-percentile of page delay for different Web cluster configurations. This figure shows that a Web cluster with less than 8 servers does not guarantee performance SLA. We observe that when the system with 4 server nodes begins to be overloaded, corresponding to 190 clients per second (cps), if we scale to 8 or more Web servers, the 90-percentile of page delay decreases of one order of magnitude, from 11.6 seconds to 1.35 seconds.

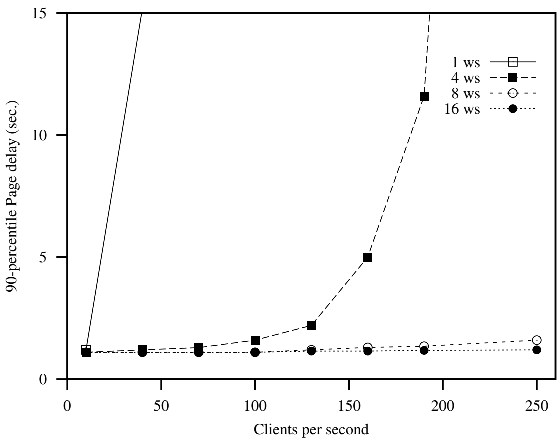

Figure 2. Static scenario: 90-percentile of page delay.

When a Web cluster with 4 nodes receives more than 160 clients per second, the system is overloaded. Figure 3 indicates that over that load threshold the peak system throughput decreases dramatically. The throughput for a Web cluster with 8 and 16 nodes continue to increase for higher numbers of client arrivals. However, the resulting throughput is much lower than the double of that related to the Web cluster with 4 nodes. The motivation is that the system with 4 nodes has a utilization much higher than that of the system with 8 and 16 nodes.

However, the main goal of Figure 3 is to demonstrate that a Web cluster with four servers requires a T3 Internet connection. When we scale-out the system to more than a certain number of servers, the Web cluster requires a larger bandwidth connection or (better) a geographically

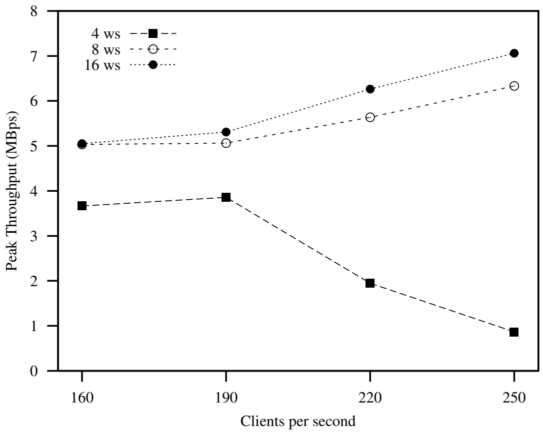

Figure 3. Static scenario: Peak system throughput.

distributed Web site. Otherwise, the risk is that the SLA is not guaranteed because of the network latency. However, we will see that passing from a locally to a geographically distributed Web system causes a relatively high loss of performance. To this purpose, in Figure 4 we compare 90-percentile of page delay at a Web cluster and at a geographically distributed Web cluster. Both architectures have the same number of server nodes and are subject to same static workload with an arrival of 400 clients per second.

To model a geographical distribution and different time zones, we divide the Internet into four world areas. Each area contains a Web cluster with four Web server nodes and various client domains. To represent the variability of traffic coming from different regions, we assign each client to one Internet region with a popularity probability that depends on the day hour in each time zone [8]. The popularity curve is taken from [1]. In the figures we consider four consecutive hours starting from 24pm until 3am. Each region is supposed to be in a time zone shifted of 6 hours from the previous region. Due to different connection popularities, in the considered four hours we have the following probabilities of receiving requests from each of the four regions: hour 24pm, 0.26, 0.08, 0.26, 0.4; hour 1am, 0.18, 0.13, 0.26, 0.43; hour 2am, 0.1, 0.17, 0.28, 0.45; hour 3am, 0.06, 0.2, 0.31, 0.43. Figure 4 evidences the difficulties of geographically architectures: at any of the four hours, the page delay is much higher than that guaranteed by the Web cluster. One motivation for this result is that request dispatching among the Web clusters is based on network proximity only that is, clients requests are assigned to the closest Web cluster. Although this policy is implemented in most real systems (e.g., [9]), the consequence is that the distributed Web cluster is highly unbalanced when the sources of traffic requests from the four regions are more skewed, say Hour 2am in our experiments. The result motivates the search for more sophisticated algorithms for geographically load balancing.

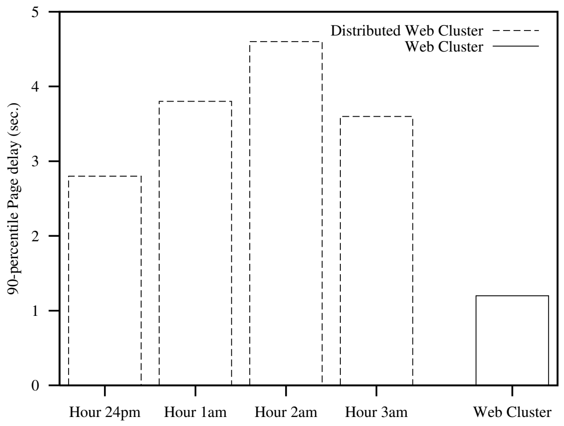

Figure 4. Web cluster vs. Distributed Web cluster.

7.2 Dynamic scenario

Dynamic requests are served through the cooperation of Web and back-end server nodes. Looking at Table 2, we can expect that in this scenario, the system bottleneck is at the back-end level. Nevertheless, in the first set of experiments we configure the Web cluster with the same number of Web server (ws) and back-end (be) nodes. The goal is to find the break-point for the system, thereby motivating an increase in the number of back-end nodes that avoids the system bottleneck.

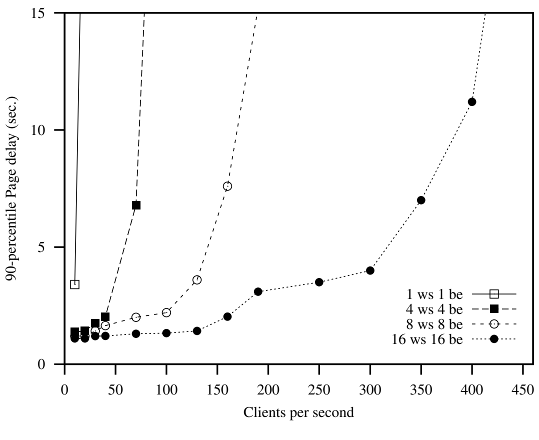

Figure 5. Dynamic scenario: 90-percentile of page delay.

Figure 5 shows that for a Web cluster with 8 servers at each level the break-point is 130 cps, when the 90-percentile of page delay is equal to 3.6 seconds. With 16 nodes at each level, we scale the break-point to 300 cps (4 seconds for page delay). This threshold is more than double than the previous limit. If we compare the results of static versus dynamic scenario, we may observe that, without an appropriate tuning of the system, performance decreases up to 50%. For example, in static scenario the break-point with 8 nodes is up to 250 cps, while in dynamic scenario the acceptable load halves that is, 130 cps. Figure 6 shows the peak throughput of the Web cluster. Analogously to the static scenario, a 16 node cluster needs a large bandwidth network or a geographically distributed architecture. For Web clusters with 4 and 8 nodes the dramatical crash of the system due to the over-utilization is evident.

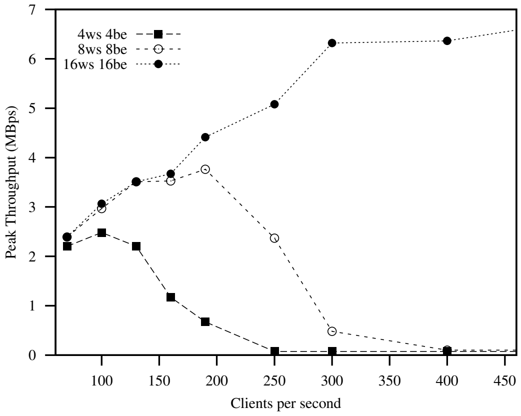

Figure 6. Dynamic scenario: Peak system throughput.

The next step is to tune the number of back-end nodes to reduce the bottleneck. In Figure 7, we start from an overloaded system with a 90-percentile of page delay of about 100 seconds and increase the number of back-end nodes until an acceptable page delay is reached. The starting point is a cluster with a number of back-end nodes (num_be) equal to the number of Web server nodes (num_ws), then we increase the ratio between num_be and num_ws from 1.5 to 5. Figure 7 shows that a cluster composed by 4 Web servers and 10 back-end nodes can manage the high workload condition, while with 8 Web server nodes we need up to 20 back-end nodes. When the ratio is over 4 Web back-end servers for each Web server, the performance begins to deteriorate because the front-end nodes become the bottleneck.

7.3 Secure scenario

In the last set of experiments we evaluate the performance of a Web system, say an e-commerce site, that is subject to a mix of static, dynamic and secure workload. As for the previous scenario, we first aim at discovering the break-point of the system and then we pass to discuss a methodology to tune the system for performance SLA. For the scenarios subject to half of secure requests, the bottleneck of the system is represented by the CPU of Web servers that must implement all secure connections and data encryption/decryption operations.

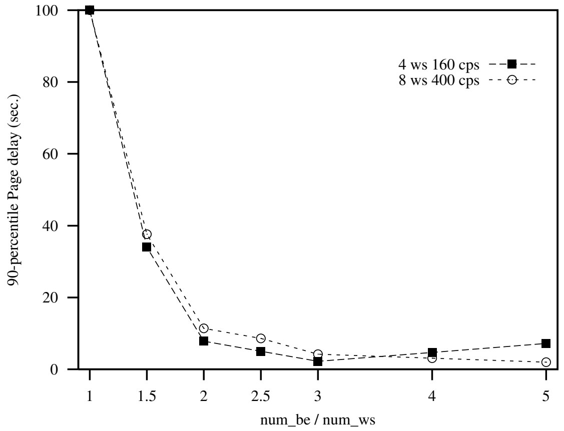

Figure 7. Dynamic scenario: 90-percentile of page delay for system tuning.

Figure 8 shows that the operations of encryption and decryption are very critical tasks. A very limited increase in client arrivals is sufficient to congestion CPU system's queues. The motivation for this critical system behavior is that each new secure session requires an authentication procedure through the computationally expensive RSA algorithm. As expected, the admitted arrival rate is about half of the load supported by a Web cluster subject to a dynamic scenario (in which half requests are not secure).

Although Figure 8 shows that at the break-point the 90-percentile of page delay is about 2 seconds, we have to consider that the setup of a new SSL session requires an exchange of 7 messages. Moreover, each successive object (if the session ID is still valid) requires an exchange of 5 messages. Hence, network delays have an impact on the page response time experimented by the client much higher than that corresponding to the previous two scenarios.

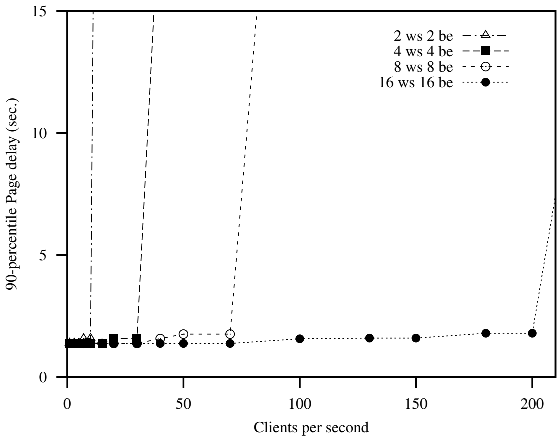

Figure 8. Secure scenario: 90-percentile of page delay.

Since for the secure scenario the bottleneck is represented by the Web server nodes, we can reduce the number of back-end nodes. To this purpose, we consider three architectures where the number of back-end nodes is a fraction of the Web server nodes number that is, 0.5, 0.75, and 1 ratios. Figure 9 shows that the best configuration is achieved for the 0.75 ratio, say 12 back-end and 16 Web server nodes. Below this ratio, the back-end nodes become the system bottleneck again.

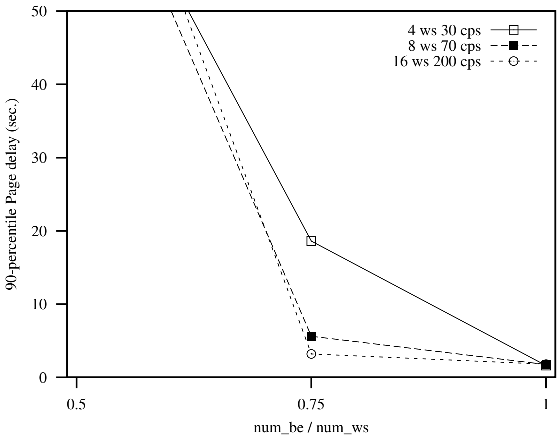

Figure 9. Secure scenario: 90-percentile of page delay for system tuning.

7.4 Significance of this performance study

From the performance study carried out in this section we can take the following main recommendations.

- To dimension and manage Web system architectures for *static scenarios* is not a big issue, even if Web sites have to serve very large files.
- When we pass to consider a *dynamic scenario*, the difficulty of choosing the right dimension of the system for SLA augments. The main reason is that a dynamic request can have a service time of two order of magnitude higher than a static request with not negligible probability. Overprovisioning of the system with respect to that required by the average load is reasonable if we want to guarantee SLA to all classes of users. In this case, the main problem is to choose the right ratio between Web server and back-end nodes. As a rule of thumb, we can reason on the basis of average service times even if mean values are not always realistic when heavy-tailed distribution functions are involved. An empiric demonstration of this result is given by Figure 7 where we see that when the dynamic load represent four-five times the static load, we need at least

two back-end servers for each Web server. At the other extreme, we have that the maximum number of Web servers per back-end is four. After this threshold, the Web server node becomes the system bottleneck.

- The *secure scenario* is the most severe. This was certainly expected, even if we were surprised to observe that even a very slight increment of the load could have crash consequences on the Web site (see Figure 8). For this reason, we conclude that Web sites that provide secure services are the only systems for which over-provisioning is highly reasonable in order to guarantee SLA.

- When we consider systems with a large number of server nodes, the network connection risks to be the system bottleneck, so we have to consider a geographically distributed Web system. Passing from a *locally* to a *geographically* distributed Web system, we have to take into account the performance loss of these latter architectures. We have seen that with present policies for geographic load balancing this loss can be extremely high. For example, if we have N server nodes in a Web cluster, we can even require $M = 3N$ servers geographically distributed to guarantee analogous SLAs. We feel that this result can be improved by using more sophisticated algorithms for geographically load balancing. However, it seems difficult to reach M/N ratios below 1.5.

8. Conclusions

This paper analyzes which locally and geographically distributed Web systems can achieve SLA for all users and services. Unlike other researches focusing on differentiated Web service approaches that favor only some classes of users and/or services, our goal is to design a distributed Web architecture that is able to guarantee the assessed SLA for all client requests. As examples of application of the analyzed systems and management policies, we consider Web sites with a mix of static, dynamic and secure requests.

References

[1] M. F. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a Web proxy in a cable modem environment. *ACM Performance Evaluation Review*, 27(2):25–36, Aug. 1999.

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

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

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

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

[7] 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.

[8] V. Cardellini, M. Colajanni, and P. S. Yu. Geographic load balancing for scalable distributed Web systems. In *Proc. IEEE Mascots 2000*, San Francisco, CA, Aug./Sept. 2000.

[9] Cisco System. DistributedDirector. `http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml`.

[10] L. Eggert and J. Heidemann. Application-level differentiated services for Web servers. *World Wide Web*, 2(3):133–142, July 1999.

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

[12] D. A. Menasce, J. Almeida, R. Fonseca, and M. A. Mendes. Resource management policies for e-commerce servers. In *Proc. Workshop on Internet Server Performance 1999*, Atlanta, GE, May 1999.

[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. 8th ACM Conf. on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, Oct. 1998.

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

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

[16] Resonate Inc. `http://www.resonate.com/`.

[17] H. Schwetman. Object-oriented simulation modeling with C++/CSIM. In *Proc. 1995 Winter Simulation Conference*, Washington, DC, Dec. 1995.

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

[19] N. Vasiliou and H. L. Lutfiyya. Providing a differentiated quality of service in a World Wide Web server. In *Proc. Performance and Architecture of Web Servers Workshop*, Santa Clara, CA, June 2000.

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