CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **22**:68–106
Published online 19 August 2009 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1470

A survey on performance management for internet applications

Jordi Guitart[1,2,*,†], Jordi Torres[1,2] and Eduard Ayguadé[1,2]

[1]*Barcelona Supercomputing Center (BSC), Barcelona, Spain*
[2]*Computer Architecture Department, Technical University of Catalonia, Barcelona, Spain*

SUMMARY

Internet applications have become indispensable for many business and personal processes, turning the performance of these applications into a key issue. For this reason, recent research has comprehensively explored mechanisms for managing the performance of these applications, with special focus on dealing with overload situations and providing QoS guarantees to clients. This paper makes a survey on the different proposals in the literature for managing Internet applications' performance. We present a complete taxonomy that characterizes and classifies these proposals into several categories including request scheduling, admission control, service differentiation, dynamic resource management, service degradation, control theoretic approaches, works using queuing models, observation-based approaches that use runtime measurements, and overall approaches combining several mechanisms. For each work, we provide a brief description in order to provide the reader with a global understanding of the research progress in this area. Copyright © 2009 John Wiley & Sons, Ltd.

Received 27 November 2008; Revised 30 April 2009; Accepted 7 June 2009

KEY WORDS: internet applications; performance management; QoS guarantees; overload control

*Correspondence to: Jordi Guitart, Jordi Girona 1-3, Campus Nord UPC, Mòdul C6, E-08034 Barcelona, Spain.
†E-mail: jguitart@ac.upc.edu

Contract/grant sponsor: Ministry of Science and Technology of Spain and the European Union (FEDER funds); contract/grant number: TIN2007-60625
Contract/grant sponsor: BSC (Barcelona Supercomputing Center)

1. INTRODUCTION

1.1. Motivation

Nowadays Internet applications are present everywhere, becoming indispensable both for people and especially for companies. This importance is translated into strong requirements in the performance, availability, or reliability of these applications. Organizations relying on Internet applications to provide services to their customers may need to be able to guarantee some quality of service (QoS) to them. In addition, they may also want to differentiate the QoS delivered to preferred customers from other ordinary customers.

This typically occurs in Internet hosting platforms (a.k.a. Internet data centers or Internet service providers), which rely on on-demand computing models (e.g. utility computing), allowing service providers to make computational resources available to customers when needed. In such a model, platform resources are shared across multiple applications in order to maximize the efficient use of the resources while minimizing the enterprise costs. Application owners pay for the actual use of platform resources, and in return, the application is provided with guarantees on resource availability and QoS, which can be expressed in the form of a service level agreement (SLA). It is obvious that in such an environment, being able to fulfill the established QoS agreements with the customers while managing the resources in the most cost-effective way is mandatory.

However, providing these performance guarantees is not straightforward. First, due to the complexity achieved in today's Internet applications. Formerly, these applications consisted of simple static web content (HTML files and images) accessed using a web browser. Nowadays, Internet applications support sophisticated web content (e.g. dynamic web content) and security capabilities to protect users' information. In addition, new web-based computing models, in which services (a.k.a. Web Services) are remotely offered over the web to the applications, have arisen too. This has led to complex architectures including several tiers (e.g. web server, application server and database server) that must interact to provide the requested service. Second, because the workload of Internet applications varies dynamically over multiple time scales (often in an unpredictable manner) and this can lead the application servers that host the services to overload (i.e. the volume of requests for content at a site temporarily exceeds the capacity for serving them). During overload conditions, the response times may grow to unacceptable levels, and exhaustion of resources may cause the server to behave erratically or even crash causing denial of services. In e-commerce applications, such server behavior could translate to sizable revenue losses. For this reason, overload prevention is a critical goal so that a system can remain operational in the presence of overload even when the incoming request rate is several times greater than the system's capacity. At the same time, it should be able to serve the maximum number of requests during such overload, while maintaining the response times (i.e. the QoS) at acceptable levels.

Taking into account these considerations, recent research has comprehensively explored mechanisms for managing the performance of Internet applications, with special focus on dealing with overload situations and providing QoS guarantees to customers. Distinct schemes have been proposed, each of them tackling the problem from a different perspective or focusing on a specific scenario. In addition to this, proposed schemes have been evolving according to the progress of Internet applications. For these reasons, great effort is required to get a global understanding of the research progress in this area. Being able to classify the proposed schemes according to a taxonomy characterizing the performance management of Internet applications would greatly improve this understanding. However, to the best of our knowledge, this effort has not been carried out in the literature yet.

1.2. Contributions

In this paper, we present a complete taxonomy that characterizes the existing schemes followed in the implementation of performance management solutions for Internet applications. Then, we perform a survey on the related literature, placing each work within the described categories and briefly describing them in order to provide the reader with a basic idea of the proposed work. In this way, the reader can realize the current research developments in this area and identify the outstanding issues. Our aim is to assist the readers to comprehend the performance management of Internet applications field by providing referral to relevant research materials. Finally, we perform a comparative discussion of the different techniques, which states their strong points and limitations.

It must be noticed that there are many works that are mentioned in several sections. This occurs when these works combine several techniques. However, the detailed description of each work is done only in the section dedicated to the technique that is more representative of the proposal.

The remainder of this paper is organized as follows: Section 2 introduces the taxonomy of techniques for the performance management of Internet applications. Section 3 presents the works using request scheduling. Section 4 introduces the proposals that use admission control and service differentiation. Section 5 describes the works focused on the dynamic resource management. Section 6 describes the approaches using service degradation. Section 7 describes the control theoretic approaches. Section 8 introduces the works based on queuing models. Section 9 describes the approaches that combine control theory and queuing models. Section 10 presents the observation-based approaches using runtime measurements. Section 11 describes the overall works that combine several techniques. Section 12 discusses about the different techniques described in this paper. Finally, Section 13 presents the conclusions of this paper and some future research directions.

2. PERFORMANCE MANAGEMENT TAXONOMY

Figure 1 shows a taxonomy for classifying the proposed techniques for dealing with the performance management of Internet applications. On one side, the techniques can be grouped depending on the actuation performed to manage the performance. Techniques in this category include request scheduling, admission control, service differentiation, dynamic resource management, service degradation, and almost any combination of them. Notice that these techniques basically cover all the actuations that a provider can undertake when an unexpected increase in an application demand jeopardizes its performance guarantees, namely allocate additional capacity to the application by assigning idle or under-used resources (i.e. dynamic resource management), degrade the performance of admitted requests (if further degradation is allowed by the agreed SLA) in order to temporarily increase the effective capacity (i.e. service degradation), or turn away excess requests (i.e. admission control), while preferentially admitting more important requests (i.e. service differentiation) and giving prior service to them (i.e. request scheduling).

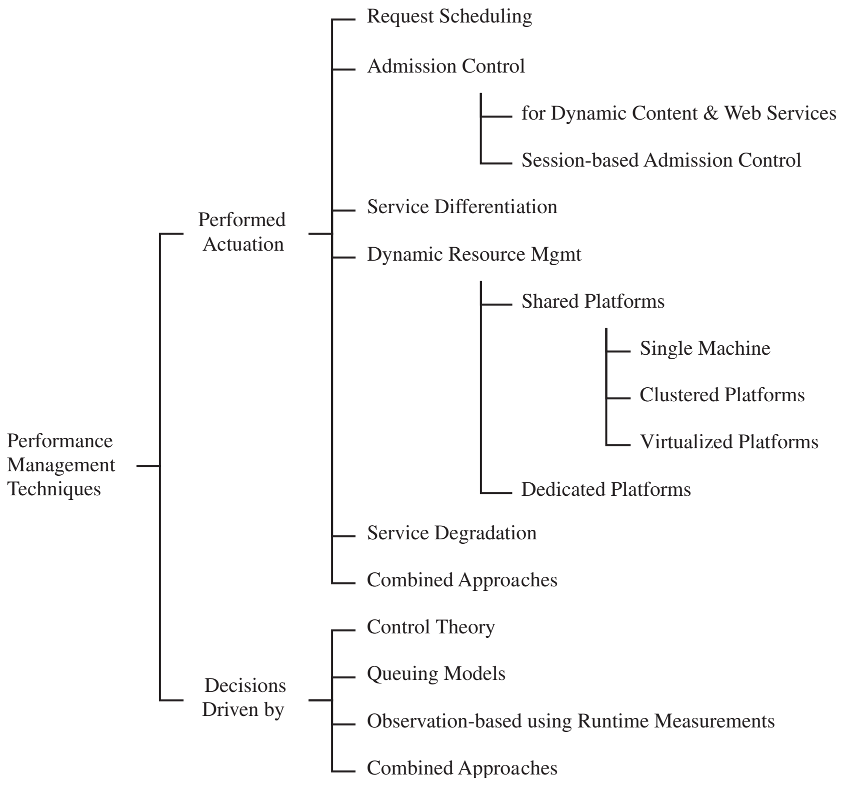

Figure 1. Taxonomy of techniques for the performance management of Internet applications.

On the other side, techniques can also be grouped depending on the mechanism used to take the performance management decisions. Based on some measurable values representing the dynamics of the system (e.g. the rate of incoming requests, the output response time, etc.), the configuration parameters of the techniques described in the previous paragraph must be computed. For instance, this includes computing the resource distribution among the different applications in the hosting platform, or computing the maximum number of requests that can be admitted. Techniques in this category include control theoretical approaches, queuing model-based approaches, observation-based approaches that use runtime measurements to make the required computations, and combinations of them.

3. REQUEST SCHEDULING

Request scheduling refers to the order in which concurrent requests should be served, as shown in Figure 2. Typically, servers have left this ordination to the operating system, which usually leads to process incoming requests in a FIFO manner. However, some proposals suggest adopting

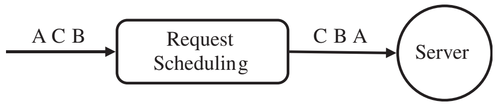

Figure 2. Request scheduling technique.

non-traditional request-ordering policies. Their common idea is to distinguish classes of requests and schedule these classes in a given order, thus providing different QoS to each class. In fact, these proposals implement the service differentiation by means of request scheduling.

For instance, several works [1–4] implement policies based on the shortest remaining processing time first (SRPT) scheduling to prioritize the service of short static web content requests in front of long requests. These works provide results that demonstrate the effectiveness of SRPT scheduling in providing significantly better response time to short requests at relatively low cost to long requests.

In particular, Crovella *et al.* [1] experiment with connection scheduling at user level, improving the mean response time by a factor of upto 4, but at the cost of a drop in throughput by a factor of almost 2. The problem is that application level scheduling does not provide fine enough control over the order in which packets enter the network. The authors evaluate their proposals by generating synthetic workloads to the Apache web server using the SURGE tool.

Balter *et al.* [2] overcome the problem in [1] by implementing the connection scheduling at kernel level (controlling the order in which socket buffers are drained into the network). This eliminates the drop in throughput and offers much larger performance improvements than [1]. However, it also requires modifying the OS kernel in order to incorporate the scheduling policy. The evaluation is conducted by issuing requests to a modified Apache web server using the S-Client software. The requests are extracted from the World Cup Soccer'98 server logs.

Schroeder and Harchol-Balter [3] demonstrate an additional benefit from performing SRPT scheduling at kernel level for static content web requests. They show that SRPT scheduling can be used to mitigate the response time effects of transient overload conditions. The authors evaluate their approach by generating a workload based on a one-day trace from the 1998 Soccer World Cup to a modified Apache web server. They use their own trace-based Web-workload generator based on the libwww library.

Finally, Rawat and Kshemkayani [4] extend the work in [2] by proposing and implementing a scheduling algorithm, which takes, in addition to the size of the request, the distance of the client from the server into account. They show that this new policy can improve the performance of large-sized files by 2.5–10%. In the evaluation, the authors use a modified version of the S-Client software to generate requests to an Apache web server. The web workload is the same as used in [2].

The size of the requests can also be used to build scheduling policies that are more sophisticated. For instance, Zhou *et al.* [5] propose a size-adaptive request-aware scheduling mechanism for busy dynamic Internet services that use the multi-threaded programming model. The authors propose managing resource usage at a request level instead of a thread or process level. In this way, the system can differentiate long requests and short requests during load peaks and prioritize resources for short requests. The scheduling algorithm, called SRQ, is implemented at kernel level and integrates size adaptiveness and deadline driven prioritization in a multiple queue scheduling framework with dynamic feedback-guided queue management. In order to evaluate their proposal, the authors have implemented SRQ in the Linux kernel and have integrated it with the Neptune clustering middleware [6]. This evaluation, which is driven by two services from Ask Jeeves search engine, demonstrates that the proposed scheduler can significantly outperform the standard Linux scheduler.

Previous works perform quite well when targeting static content web requests. However, they are not suitable for being used with Internet applications based on dynamic content web requests or web services. These applications generally benefit from taking into account the business value of the requests for ordering them. According to this, some works [7–10] implement policies that schedule the requests depending on their priority. In these works, the priority of a request is assigned considering business parameters, such as the type of customer that has issued the request (as done in [7,8]) or the reward that is likely to generate the request (as done in [9,10]).

In particular, Almeida *et al.* [7] propose priority-based request scheduling as a mechanism for providing differentiated QoS. Priorities to requests are assigned based on the customer to whom the requested file pertains. The authors implement the priority-based scheduling at both user and kernel levels. In the user-level approach, the Apache web server is modified with the inclusion of a scheduler process responsible for deciding the order in which the requests should be handled. In the kernel-level approach, the Linux kernel is modified such that requests priorities are mapped into priorities of the HTTP processes handling them. The evaluation shows upto 26% of improvement for higher-priority requests, with an accompanying 504% fall in the performance of lower ones, for the user-level approach. For the kernel-level approach, improvement is similar, while the slowdown is around 208%. The web workload used in the evaluation is generated using the WebStone benchmark.

Alonso *et al.* [8] propose a mechanism to provide differentiated QoS to the different client categories by assigning different priorities to the threads attending the connections, depending on the type of client. The authors propose to schedule threads using native operating system priorities arguing that priorities at the JVM level are not considered for the scheduling of threads at the kernel level. The authors provide an implementation of the mechanism using Linux Real Time priorities. The evaluation of this proposal, which is conducted by using the httperf workload generator to establish SSL connections with a Tomcat server for requesting static web content, shows that higher-priority clients have better QoS, especially in high competition situations.

Menasce *et al.* [9] consider the problem of resource scheduling in e-commerce sites with the aim of maximizing the revenue. The authors describe customers' sessions with a customer behavior model graph (CBMG) and then propose a priority scheme for scheduling requests at the user level based on the states of the users. Priorities change dynamically as a function of the state, a customer is in and as a function of the amount of money that the customer has accumulated in the shopping cart. In order to evaluate the proposed policy, the authors have built an e-commerce site simulator, which is a hybrid between a trace-driven simulator and a discrete event simulator. SURGE is used to generate requests to start the sessions at the e-commerce site. Simulation results show that the suggested scheme can increase the revenue as much as 29% over the no priority policy for an overloaded site.

Finally, Totok and Karamcheti [10] propose a user-level request scheduling mechanism, called reward-driven request prioritization (RDRP), which maximizes the reward attained by an Internet service by dynamically assigning higher execution priorities to the requests whose sessions are likely to bring more profit (or any other application-specific reward) to the service. The authors use a Bayesian inference analysis to statistically predict future sessions structure (in the form of a

Table I. Characteristics of works including request scheduling in their solution.

work	policy	level	joint?	workload
[1]	SRPT	user	no	Surge (static) on Apache
[2]	SRPT	kernel	no	World Cup Soccer'98 logs (static) on Apache
[3]	SRPT	kernel	no	World Cup Soccer'98 logs (static) on Apache
[4]	SRPT	kernel	no	World Cup Soccer'98 logs (static) on Apache
[5]	SRQ	kernel	no	Ask Jeeves search on Neptune
[7]	priority (type of client)	both	no	WebStone (static) on Apache
[8]	priority (type of client)	kernel	no	static & SSL on Tomcat
[9]	priority (reward)	user	no	simulation of e-commerce site (dynamic)
[10]	RDRP (priority, reward)	user	no	TPC-W (dynamic) on JBoss
[11]	priority (type of client)	user	yes	8kb static web pages on Apache
[12]	priority (reward) + GPS	user	yes	Matlab simulation of electronic store (dynamic)
[13]	DWFS (priority, session completion probability)	user	yes	WebStone (static) on Apache
[14]	SJF	user	yes	TPC-W (dynamic) on Tomcat
[15]	weighted round robin for each traffic class	user	yes	Trade 6 benchmark (dynamic) on IBM WebSphere
[6]	priority (service yield)	user	yes	Ask Jeeves search on Neptune
[16]	SRJF	user	yes	simulation of messaging service based on JMS
[17]	priority (type of client)	kernel	yes	WebStone (static & CGI files) on Apache

CBMG) by comparing their requests seen so far with aggregated information about client behavior. The predicted reward and execution cost values are used to compute each request priority, which is used in scheduling bottleneck server resources. The authors have integrated their proposed methods in the open-source application server JBoss, and they have used the TPC-W transactional web e-commerce benchmark to conduct the evaluation, showing that RDRP techniques yield benefits in both underload and overload situations, for both smooth and bursty client behavior.

Table I summarizes the main characteristics (concerning this technique) of the works using request scheduling in their solution. This includes the scheduling policy, the level where this policy is implemented (i.e. user, kernel or both), whether this work uses other techniques in addition to scheduling[‡], and its target workload.

However, although request scheduling can improve the response times, under extreme overloads other mechanisms become indispensable. Anyway, better scheduling can always be complementary to any other mechanism. In this sense, some works have incorporated request scheduling to their overload prevention solutions [6,11–17]. Detailed description of these works can be found in the corresponding sections.

4. ADMISSION CONTROL AND SERVICE DIFFERENTIATION

Service differentiation is based on differentiating classes of customers and providing different QoS to each class, as shown in Figure 4. This allows, for instance, maintaining response times

[‡]Further details on the techniques used can be found in Table VII.

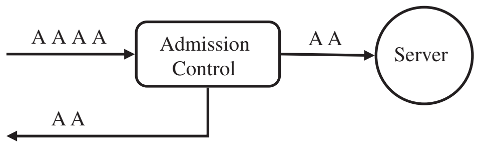

Figure 3. Admission control technique.

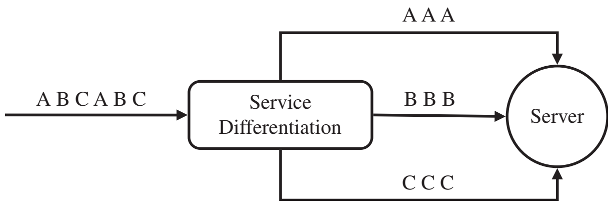

Figure 4. Service differentiation technique.

of preferred clients in the presence of overload. As discussed in the previous section, service differentiation can be implemented by means of scheduling. Other approaches, such as refusing connections coming from a given customer's class, allocating more resources to higher-priority services or establishing different target delays for different service classes, are introduced in this section and in the following ones.

Admission control is based on reducing the amount of work, the server accepts when it is faced with overload by refusing a percentage of connections, as shown in Figure 3. Simpler admission control approaches refuse all the incoming connections when predefined thresholds in the system (e.g. the number of pending connections in the queue or the CPU utilization) are exceeded [18]. Nevertheless, admission control is generally approached as a special case of service differentiation, where a given customer's class (typically the lower-priority class) is provided with a 'refused' level of service when the server is overloaded. For this reason, admission control and service differentiation have been combined in a great amount of works [11,17,19–24] to prevent server overload and provide differentiated QoS to clients.

For instance, Bhatti and Friedrich [11] propose WebQoS, an architecture for web servers to provide QoS to differentiated clients. The authors use request classification, admission control, and request scheduling for supporting distinct performance levels for different classes of users and maintaining the predictable performance even when the server is overloaded. Requests are classified into priorities according to the customers' preference, and admission control of low-priority requests is triggered when thresholds in the number of requests queued and the number of premium requests queued are exceeded. Admitted requests are scheduled for execution depending on the selected policy. The authors suggest several potential scheduling policies at the user level, but evaluate only priority scheduling. Nevertheless, they do not experimentally demonstrate sustained throughput in the presence of overload. This work considers only static web content by providing an example implementation using Apache. The client workload is generated using the httperf tool.

Chen *et al.* [19] present an admission control algorithm called admission control based on estimation of service time that attempts to limit the number of admitted requests based on estimated service times. Admission of a request is decided by comparing the available computation power for a duration equivalent to the predetermined delay bound with the estimated service time of the request. A double-queue structure is used to handle the over/under admission problems due to bad estimation of service times. The authors also present an extension to give better service to higher-priority tasks. The available resources for lower-priority tasks are equal to the system capacity minus the predicted resource requirements for higher-priority tasks. In order to test the effectiveness of their proposal, the authors have developed an event-driven simulator and use real traces of web requests (extracted from the logs of the CS departmental web server at Michigan State University) as input.

Iyer *et al.* [18] propose three simple schemes for web server overload control. The first scheme uses thresholds on the connection queue length for selectively dropping incoming requests as they arrive at the server. The second scheme provides feedback to the proxy during overloads, which would cause it to restrict the traffic being forwarded to the server. The third scheme is simply a combination of the other two schemes. The evaluation of this proposal is conducted by requesting a single ASP file to an IIS server. Although the evaluation results show that these schemes improve the server throughput under overload, the authors do not address how the thresholds may be determined online.

Jamjoom and Reumann [20] propose an adaptive mechanism called QGuard that uses traffic shaping to provide the overload protection and the differential service for Internet servers. QGuard drives the selection of incoming packet rates based on the observation of different system load indicators. However, it cannot take the potential resource consumption of requests into account, having to reduce the acceptance rate of all requests when one resource is over-utilized. For evaluating their mechanism, the authors have implemented a load-generating server application that can be configured to generate different types of load: CPU activity, memory usage, file accesses, and the generation of large response messages.

Kanodia and Knightly [21] present a queuing-based algorithm, called latency-targeted multiclass admission control, for admission control and service differentiation using request and service statistical envelopes (a modeling technique used to characterize the traffic of multiple flows over a shared link). The authors propose a server architecture having one request queue per service class. The admission controller attempts to meet latency bounds for the multiple classes using measured request and service rates of each class. The algorithm is only evaluated via trace-driven simulation using streams of tokenized requests collected from logs of two real Web servers: the server of the Computer Science Department at Rice University and the server for the 1998 World Cup Soccer.

Li and Jamin [22] describe a measurement-based admission control algorithm to provide the proportional bandwidth allocation to clients independently of the amount of requests generated by each one. Bandwidth percentage for each client is statically specified at server startup. Requests from a client are rejected when it has both received more than its allocated bandwidth and the server is fully utilized. This approach considers the introduction of controlled amounts of delay in the processing of certain requests during overloads to ensure the different classes of requests are receiving the appropriate share of the bandwidth. The authors evaluate their proposal on the Apache web server using the httperf measurement tool.

Voigt *et al.* [17] present different kernel-based mechanisms for admission control and service differentiation in overloaded web servers. Their mechanisms include TCP SYN policing, prioritized listen queue, and HTTP header-based connection control. TCP SYN policing limits the number of incoming connection requests using a token bucket policy and prevents overload by enforcing a maximum acceptance rate of non-preferred clients. Prioritized listen queue reorders the listen queue based on the predefined connection priorities (request URL and client IP address) in order to provide low delay and high throughput to clients with high priority. Finally, HTTP header-based connection control provides admission control and priority based on the application-level information such as URLs and cookies. The authors have implemented the proposed kernel mechanisms in the AIX 5.0 system, and have evaluated the performance of a modified Apache Web server using WebStone 2.5 and a modified version of S-Client as workload generators.

Finally, Voigt and Gunningberg [23] present an adaptive architecture that performs admission control based on the expected resource consumption of requests in order to avoid over-utilization of the critical resources. The authors identify the resource intensive requests and the resource they demand by the URL in the HTTP header and adapt the rate of CPU and bandwidth intensive requests according to the utilization of the corresponding resource. The adaptation of the acceptance rates is done using feedback control loops. This proposal is evaluated on the Apache web server using the S-Client tool for generating a workload that is derived from the SURGE traffic generator.

4.1. Admission control for dynamic content sites and web services

The aforementioned works on admission control are based on web systems serving only static content, and for this reason, most of the proposed solutions are not directly applicable on multi-tiered sites based on the dynamic web content or web services. The main challenge in these environments is to determine how the acceptation of a new request would affect the state of the system. Formerly, it was assumed that request cost was linear in proportion to the size of the response generated. While this is true for static web content, the times of serving dynamic content or web services have much greater variability, which has no relation to the size of the response generated. Because of this, recently some works [14,16,25,26] have focused specifically on proposing admission control and service differentiation solutions for these systems.

For instance, Elnikety *et al.* [14] present a proxy called Gatekeeper that transparently performs admission control and user-level request scheduling for multiply tiered e-commerce Web sites by externally observing execution costs of requests. When a request arrives to the server, Gatekeeper determines if admitting the request will exceed the capacity of the system by using an estimation of the resource usage for that request. In addition, Gatekeeper sorts the admission queue based on their expected processing times (SJF scheduling). Since the proxy is external, no modification to the host OS, web server, application server or database is required. However, this proposal depends on offline profiling in order to determine the capacity of the system. The evaluation is conducted by using the TPC-W benchmark for generating an e-commerce workload to a Tomcat application server.

Guitart *et al.* [25] present a session-based admission control mechanism for secure environments, which is based on SSL connections differentiation depending on if the SSL connection reuses an existing SSL connection on the server or not. This mechanism prioritizes the acceptation of client connections that resume an existing SSL session, in order to increase the probability for a client to complete a session, maximizing in this way the number of sessions successfully completed. This allows e-commerce sites based on SSL to increase the number of transactions completed, thus generating more revenue. In addition, the mechanism prevents applications overload by limiting the acceptation of new SSL connections to the maximum number acceptable by the application without overloading, depending on the available computation power and the estimated service time of the incoming connections. This proposal is evaluated by using the httperf generator to issue secure requests to a Tomcat server with the RUBiS benchmark deployed.

Verma and Ghosal [16] propose a service time-based online admission control methodology for maximizing the profits of a service provider. The authors derive an online version of the shortest remaining job first (SRJF) policy for taking the admission control decision of a request. They use an estimate of the request service time, its QoS bounds, the prediction of arrivals and services times of requests to come in the short-term future, the rewards associated with servicing a request within its QoS bounds, and the capacity availability (after accounting for the usage of already admitted requests). The admission control admits a subset of requests that would maximize the profit of the provider. The evaluation is carried out with a content-based publish/subscribe messaging middleware service based on the standard Java Messaging Service (JMS).

Finally, Welsh and Culler [26] describe an adaptive approach to overload control in the context of the SEDA web server [27]. SEDA decomposes services into a set of event-driven stages connected with request queues. Each stage can perform admission control based on monitoring the response time through the stage. The admission is controlled by using an additive-increase multiplicative-decrease algorithm that works by gradually increasing admission rate when the performance is satisfactory and decreasing it multiplicatively upon observing QoS violations. Their solution relies mainly on a heuristic approach for controller design (control parameters are set by hand after running tests against the system) instead of using control-theoretic techniques. The downside of this approach is that a request may be rejected late in the processing pipeline, after it has consumed significant resources in the previous stages. The evaluation includes dynamic content in the form of a web-based email service (Arashi). Emulated users access the service based on a simple Markovian model of the user behavior derived from traces of the UC Berkeley Computer Science Division's IMAP server.

4.2. Session-based admission control

In many web sites, especially in e-commerce, most of the applications are session-based. A session contains temporally and logically related request sequences from the same client. Session integrity is a critical metric in e-commerce. For an online retailer, the higher the number of sessions completed, the higher the amount of revenue that is likely to be generated. The same statement cannot be made about the individual request completions. Sessions that are broken or delayed at some critical stages, like checkout and shipping, could mean loss of revenue to the web site.

Sessions have distinguishable features from individual requests that complicate the overload control. For this reason, admission control techniques that work on a per request basis, such as limiting the number of threads in the server or suspending the listener thread during overload periods, may lead to a large number of broken or incomplete sessions when the system is overloaded (despite the fact that they can help to prevent the server overload). Most of the aforesaid admission control works suffer from this limitation when dealing with session-based workloads. This has motivated some proposals [12,13,25,28,29] to prevent overload considering the particularities of session-based applications.

For instance, Carlstrom and Rom [12] describe an architecture for request scheduling at user level and session-based admission control in web servers. The admission controller is rate-based. New sessions arriving when the maximum admitted session arrival rate has been achieved will be refused. If the initial request of a session is admitted, all subsequent requests within the same session will be also admitted. Once the session has been established, each request is classified with respect to the requested stage in the session, and enters in a stage-specific FIFO queue before receiving service. When the critical resource is freed, queued requests are scheduled using a generalized processor sharing (GPS) algorithm that uses techniques for nonlinear optimization (based on steady-state queue behavior) for maximizing an application-specific reward function. The evaluation is based on simulating an electronic store using Matlab.

Chen and Mohapatra [13] characterize a commercial web server log for deriving session-based dependency relationships among HTTP requests. Based on the session-level traffic model, the authors propose a dynamic weighted fair sharing (DWFS) scheduling algorithm to control overload. This work combines an admission control mechanism (proposed by the authors in a previous work [19]) that admits as many sessions as possible so long as the server is not overloaded with the DWFS algorithm that discriminates the scheduling of requests based on the probability of completion of the session that the requests belong to. In this way, requests of sessions that have a higher probability of being completed are scheduled first. The authors evaluate their proposal over an Apache web server using a modified version of WebStone 2.5.

Cherkasova and Phaal [28] propose a session-based admission control scheme and demonstrate that considering session characteristics in admission control reduces the number of sessions rejected. This approach allows admitted sessions to run to completion even under overload, denying new sessions when the observed server utilization exceeds a predefined threshold. Once the observed utilization drops below the given threshold, the server begins to admit and process new sessions again. The authors evaluate their proposal by simulating a commercial site and using a simple model to characterize sessions.

Finally, Voigt and Gunningberg [29] extend the architecture proposed in their previous work [17] to provide kernel-based control of persistent connections. The goal is to allow the important sessions (understanding sessions as the sequence of individual requests on the same persistent connection) to complete when the server becomes overloaded. In this approach, the importance of persistent connections is based on the cookies found in the HTTP header. When the CPU utilization is higher than a predefined threshold, they do not abort all the incoming connections, but only the ones that are considered less important (those do not carrying a suitable cookie). However, this approach relies on the existence of unimportant persistent connections that can be aborted. If all the active persistent connections are important, the system will not abort any of them and will fail to recover. This proposal is evaluated on the Apache Web server using a modified version of the S-Client tool. The workload consists of static and dynamic requests. The latter are minor modifications of WebStone CGI files.

Table II summarizes the main characteristics (concerning this technique) of the works using admission control and service differentiation in their solution. This includes how the admission control mechanism is triggered, the actuation performed, whether this work supports sessions, whether it also uses other techniques, and its target workload.

Table II. Characteristics of works including admission control and service differentiation in their solution.

work	triggered by	actuation	sess?	joint?	workload
[11]	threshold in queue length (total and premium)	rejection of basic requests	no	yes	8kb static web pages on Apache
[19]	available computation power vs. estimated service time	rejection of requests (lowest priority first)	no	no	event-driven simulation of MSU CS server logs (static)
[18]	threshold in queue length	rejection of requests	no	no	single ASP file on IIS server
[20]	system load indicators	traffic shaping (rate)	no	no	own server that can be configured to generate different types of load
[21]	request & service rate per class	rejection of requests per class	no	no	trace-driven simulation of Rice server & World Cup Soccer’98 logs (static)
[22]	threshold in bandwidth per client	rejection of requests per client	no	no	10kb static web pages on Apache
[17]	token bucket policy defined by rate & burst	rejection of requests	no	yes	WebStone (static & CGI files) on Apache
[23]	CPU utilization & queue length	rate adaptation	no	no	Surge (static) & CGI files on Apache
[24]	threshold in stretch factor per class (response time/service demand)	rejection of requests (lowest priority class first)	no	yes	Internet search & online auction (dynamic)
[14]	estimated resource usage of requests vs. system capacity	rejection of request	no	yes	TPC-W (dynamic) on Tomcat
[25]	available computation power vs. estimated service time	rejection of new clients	yes	no	RUBiS (dynamic) on Tomcat
[16]	estimated service time & associated reward vs. capacity availability & predicted arrivals	rejection of request	no	yes	simulation of messaging service based on JMS
[26]	additive-increase multiplicative-decrease algorithm on 90th-percentile response time	rejection of requests per stage	no	no	Arashi web-based email service (dynamic) on SEDA
[12]	rate threshold based on expected reward	rejection of new session	yes	yes	Matlab simulation of electronic store (dynamic)
[13]	available computation power vs. estimated service time	rejection of requests (lowest priority first)	yes	yes	WebStone (static) on Apache
[28]	threshold in utilization	reject new sessions	yes	no	commercial site simulation
[29]	threshold in CPU utilization	reject connections without proper cookie	yes	no	12–29 kb static files & CGI files on Apache

Table II. *Continued.*

work	triggered by	actuation	sess?	joint?	workload
[30]	serving request before its maximum allowed response time unfeasible	selective request dropping	no	no	TPC-W & Teoma Internet Search Engine (dynamic) on Apache/Tomcat/Neptune
[31]	deviation from target settings in CPU & memory	update MaxClients & KeepAlive Timeout	no	no	WebStone (static) on Apache
[32]	deviation from target response time	adjust request acceptance probability	no	yes	TPC-W (dynamic) on Tomcat
[33]	deviation from target response time	adjust request acceptance probability	no	yes	TPC-W (dynamic) on Tomcat
[34]	available computation power vs. estimated service time	rejection of new clients	yes	yes	RUBiS (dynamic) on Tomcat
[35]	serving request before its maximum allowed response time unfeasible	rejection of requests within a batch	no	yes	1–256 kb static files & PHP scripts on Apache

While admission control and service differentiation can be effective for providing differentiated QoS to clients and preventing server overload, they must be used with caution. Using admission control determines that a percentage of requests are refused, which can translate into unhappy clients. For this reason, dropping too many requests should be prevented since this will also cause revenue loss. According to this, it would be desirable to attempt less aggressive solutions before. In addition, session-based workloads require specialized admission control solutions to support performance guarantees. Furthermore, most of the proposed solutions trigger admission control when some given thresholds are surpassed. In general, these thresholds are static and must be determined beforehand. For this reason, these solutions are not directly applicable if we aim for an autonomic system.

5. DYNAMIC RESOURCE MANAGEMENT

As commented before, the hosting platform is responsible for providing their running applications with enough resources to guarantee the agreed QoS in the SLA. When the hosting platform fails to fulfill these QoS guarantees, it must compensate the customer by paying him a penalty. In order to avoid these payments, the hosting platforms tend to over-provision the applications with an amount of resources according to their highest expected resource demand. However, as the resource demand of Internet applications can vary considerably over time, this can lead to resource under-utilization in the hosting platform most of the time. Of course, this is not desirable when a profitable hosting platform is envisioned. Notice that the unused resources could be temporarily allocated to another service, improving in this way the utilization and the profit for the provider.

Taking this into account, recent studies [36–38] have reported the considerable benefits of dynamically reallocating resources among hosted applications based on the variations in their workloads instead of statically over-provisioning resources in a hosting platform. The goal is to meet the applications' requirements on demand and adapt to their changing resource needs, as shown in Figure 5. In this way, better resource utilization can be achieved and the system can react to unexpected workload increases. In addition, differentiated service can be easily provided to different customers' classes by varying their allocated resources.

This premise has been used as the basis of several works, which dynamically provision resources to applications on demand in order to meet the agreed QoS while using the hosting platform resources in a cost-effective way. These works can be classified depending on the resource provisioning model they consider, which can be either a dedicated or a shared model (see [36]). In the dedicated model, some cluster nodes are dedicated to each application and the provisioning

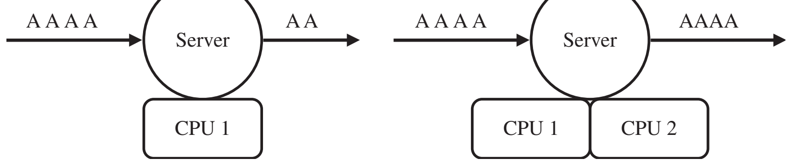

Figure 5. Dynamic resource management technique.

technique must determine how many nodes to allocate to the application. In the shared model, the node resources can be shared among multiple applications and the provisioning technique needs to determine how to partition the resources on each node among competing applications.

5.1. Resource management in dedicated hosting platforms

Some works [35,39–43] have addressed the resource provisioning problem using the dedicated model. For instance, Appleby *et al.* [39] present Oceano, a dedicated management infrastructure for server farms developed at IBM. In Oceano, the server farm is physically partitioned into clusters, each serving one customer. The servers can be moved dynamically across clusters depending on the customers' changing needs. The addition and removal of servers from clusters is triggered by SLA violations triggered by monitoring agents. This approach suffers of lack of responsiveness due to the latency of the server transfer operation from one cluster to another.

Bouchenak *et al.* [41] present Jade, a middleware for self-management of distributed software environments. The main principle is to wrap legacy software pieces in components in order to provide a uniform management interface, thus allowing the implementation of management applications. The authors use Jade to implement a self-optimizing version of an emulated electronic auction site deployed on a dedicated J2EE cluster. The self-optimization, which is implemented using a control loop based on thresholds, consists of dynamically increasing or decreasing the number of replicas (at any tier of the J2EE architecture) in order to accommodate load peaks. The proposed self-optimization is very simple, intended only to demonstrate the ability of Jade to implement specific autonomic components. The evaluation has been realized by deploying the RUBiS benchmark on the Tomcat application server.

Finally, Ranjan *et al.* [42] describe a framework for QoS-driven dynamic resource allocation in dedicated Internet data centers. The authors propose a simple adaptive algorithm to reduce the average number of servers used by an application while satisfying its QoS objectives. The algorithm assumes a G/G/N open queuing model with FCFS scheduling on each server in the cluster, and response time linearly related to utilization. This proposal is evaluated with a trace-driven simulator based on YacSim using traces from large-scale e-commerce and search-engine sites (i.e. google).

Nevertheless, hosting platforms based on a dedicated model use to be expensive in terms of space, power, cooling, and cost. For this reason, shared model constitutes an attractive lower cost choice for hosting environments when dedicated model cannot be afforded.

5.2. Resource management in shared hosting platforms

5.2.1. Resource management in a single machine

In the context of resource management solutions using the shared model, some works [44,45] have developed mechanisms for predictable allocation of resources in single machine environments.

In particular, Banga *et al.* [44] provide resource containers, a new operating system abstraction that embodies a resource enabling fine-grained allocation of resources and accurate accounting of resource consumption in web server-like applications. When combined with proportional resource schedulers, they enable the provision of differentiated services, where a predetermined fraction of the server resources is guaranteed to be available to each service. The authors have prototyped their proposal modifying a Digital Unix kernel. The evaluation is conducted by issuing requests to a modified version of the thttpd server (changed in order to use resource containers) from a number of clients running the S-Client software.

Finally, Liu *et al.* [45] describe the design of online feedback control algorithms to dynamically adjust entitlement values for a resource container on a server shared by multiple applications. The goal is to determine the minimum level of entitlement for the container such that its hosted application achieves the desired performance levels. A proportional integral (PI) controller is designed offline for a fixed workload and a self-tuning adaptive controller is described to handle limited variations in the workload. These controllers are used to compute the proper level of CPU entitlement for the Web server in order to maintain the measured mean response time around the desired response time. The controllers are implemented and evaluated on a testbed using the HP-UX PRM as the resource container and the Apache web server as the hosted application in the container. Client requests are generated using the httperf tool.

5.2.2. *Resource management in clustered hosting platforms*

Whereas the mechanisms for resource management in a single machine are useful, they are insufficient in typical hosting platforms, which are generally comprised of clusters of nodes. These systems require coordinated resource management mechanisms that consider all the nodes in the cluster. According to this, addressing the resource provisioning problem for clustered architectures using the shared model has been the main goal of numerous works [6,24,46–52].

In particular, Aron *et al.* [46] present Cluster Reserves, a resource management facility for cluster-based web servers that afford effective performance isolation in the presence of multiple services that share a server cluster. Cluster Reserves extend single-node resource management mechanisms (i.e. Resource Containers [44]) to a cluster environment. This work provides differential service to clients by providing fixed resource shares to services spanned in multiple nodes and dynamically adjusting the shares on each individual server to bound aggregate usage for each hosted service based on the local resource usage. The authors employ a linear programming formulation for allocating resources, resulting in polynomial time complexity. The authors evaluate their proposal by running the Apache Web server at the server nodes and generating requests with a client program based on the S-Client software.

Chase *et al.* [49] present the design and implementation of Muse, which is an architecture for resource management in a shared hosting center with an emphasis on energy as a driving resource management issue. Muse uses an economic model for dynamic provisioning of resources in which services bid for resources as a function of delivered performance. The system continuously monitors load and plans resource allotments by estimating their effect on service performance. A greedy resource allocation algorithm, called maximize service revenue and profit, adjusts resource prices to balance supply and demand. A major goal of this work is to maximize the number of unused servers so that they can be powered down to reduce the energy consumption. The authors evaluate their approach on Apache Web servers by generating combined workloads of synthetic traffic and real request traces with the SURGE workload generation tool.

Chase *et al.* [50] describe the architecture and implementation of cluster-on-demand, an automated framework to manage resources in a shared hosting platform, which allows partitioning on-the-fly a physical cluster into multiple independent virtual clusters. A virtual cluster (vcluster) is a functionally isolated subset of cluster nodes configured for a common purpose, with associated user accounts and storage resources, a user-specified software environment, and a private IP address block and DNS naming domain. Nodes can be reallocated among these virtual clusters as needed (according to demand or resource availability), reconfiguring them as needed with PXE network boots. The system resizes each dynamic vcluster in cooperation with the service hosted within it, which contains the logic for monitoring load and changing membership in the active server set. The evaluation uses the Sun's GridEngine and traces from the Duke Computer Science server.

Shen *et al.* [6] present an integrated resource management framework (part of Neptune system) that provides flexible service quality specification, efficient resource utilization, and service differentiation for cluster-based services. The authors introduce a unified quality-aware metric, called 'quality-aware service yield', which depends on the response time. The overall goal of the system is to maximize the aggregate service yield resulting from all requests. The resources are managed through a two-level request distribution and scheduling scheme. At the cluster level, requests for each service class are evenly distributed to all replicated service nodes without explicit partitioning. Inside each node, an adaptive scheduling policy adjusts to the runtime load condition and seeks high aggregate service yield. This policy considers the relative deadline of the requests, their expected resource consumption (estimated using an EWMA filter), and their expected yield. The proposed framework is evaluated using traces of two service workloads. The first service is a Differentiated Search service based on an index search component from Ask Jeeves search. The second service, called Micro-benchmark, is based on a CPU-spinning benchmark.

Urgaonkar *et al.* [51] present techniques for provisioning CPU and network resources in shared hosting platforms running potentially antagonistic third-party applications. The authors first derive the resource usage profiles of services using kernel-based resource monitoring on a dedicated node running the service, and use these profiles to guide the placement of application components onto shared nodes. Then they propose techniques to overbook the cluster resources in a controlled manner such that the platform can maximize its revenue while providing probabilistic QoS guarantees to applications (resources allocated to services correspond to a high percentile of the resources needed to satisfy all requests). However, the system proposed does not describe any continuous monitoring facility or reaction mechanism (resource usages are only measured when deriving profiles). In addition, provisioning decisions are based on the application steady state instead of considering rapidly changing demands. The proposed techniques are implemented in a Linux kernel and evaluated using an Apache web server with the SpecWeb99 benchmark, an MPEG streaming media server, a Quake game server, and a PostgreSQL database server with the pgbench benchmark.

Finally, Urgaonkar and Shenoy [52] present the design of Sharc, a system that enables resource sharing among applications running on a shared hosting platform. Sharc extends the benefits of resource management mechanisms in a single node, such as reservations or shares to clustered environments. The authors present techniques for managing CPU and network interface bandwidth on a cluster-wide basis, supporting resource reservations based on the applications resource requests (made at their startup time), dynamic resource allocation based on applications past resource usage, and performance isolation of applications. This proposal is evaluated using two types of workloads: a commercial third-party hosting platform workload (an e-commerce application, a replicated Apache web server, a file download server, and a home-grown streaming media server) and a research workgroup environment workload (a compute-intensive scientific application, an information retrieval application, a compute-intensive disk simulator, and an application build job).

5.2.3. Resource management in virtualized hosting platforms

Recently, the use of virtualization has been explored for cost reduction and easier management in shared hosting platforms. Virtualization allows the consolidation of services, multiplexing them onto physical resources while supporting their isolation from other services sharing the same physical resource, reducing in this way providers costs and maintaining the integrity of the underlying resources and the other services. Virtualization has other valuable features for hosting platforms. It offers the image of a dedicated and customized machine to each user, decoupling them from the system software of the underlying resource. In addition, virtualization allows agile and fine-grain dynamic resource provisioning by providing a mechanism for carefully controlling how and when the resources are used for migrating a running machine from resource to resource if needed. According to this, recently several works [53–58] have proposed resource management solutions over virtualized platforms.

In particular, Govindan *et al.* [55] have developed a new communication-aware CPU scheduling algorithm and a CPU usage accounting mechanism to address the existing obstacles in the efficient operation of highly consolidated virtualization-based hosting platforms. The CPU scheduling algorithm incorporates the I/O behavior of the overlying VMs into its decision-making. The key idea is to introduce short-term unfairness in CPU allocations by preferentially scheduling communication-oriented applications over their CPU-intensive counterparts. The CPU overheads resulting from server virtualization are also incorporated into the CPU scheduling algorithm. The evaluation is conducted in a Xen-based hosting platform using two applications: TPC-W-NYU, which is a three-tiered application based on the TPC-W benchmark for an online bookstore, deployed on a JBoss server, and a streaming media server written in Java.

Norris *et al.* [56] propose OnCall, a spike management system based on virtual machines (i.e. VMware GSX server) for shared hosting clusters that serve dynamic content. The system relies on an economically efficient marketplace for computing resources, allowing applications to trade computing capacity on the market using automated market policies. In this way, OnCall can multiplex several (possibly competing) applications onto a single server cluster and provide enough aggregate capacity to handle temporary workload surges for a particular application while guaranteeing some capacity to each application in steady state. The proposed system is evaluated using simulation. Load is generated using the Apache JMeter testing tool.

Finally, Xu *et al.* [58] present a two-level autonomic resource management system that enables automatic and adaptive resource provisioning in accordance with SLA specifying dynamic tradeoffs of service quality and cost. The authors implement local controllers at the virtual-container level and a global controller at the resource-pool level. A novelty of the controller designs is their use of fuzzy logic to characterize the relationship between application workload and resource demand. The evaluation is conducted on a data center virtualized with VMware ESX Server, using a combination of e-business applications (e.g. Java Pet Store) and traces collected from the ’98 World Cup site. Java Pet Store requests are generated with the httperf tool, while the ’98 World Cup workload is replayed using the Real-Time Web Log Replayer application.

Table III summarizes the main characteristics (concerning this technique) of the works using dynamic resource management in their solution. This includes the mechanism, which drives the resource allocation decisions, the resource provisioning model (i.e. shared (single machine (S), clustered platform (C), virtualized platform (V)) or dedicated), whether this work uses also other techniques, and its target workload.

Dynamic resource management has become an indispensable technique for providing performance guarantees to customers, and at the same time, fully exploiting the platform resources. Having this in mind, shared platforms (and in particular those using virtualization) have consolidated as the best choice. However, to obtain better results, dynamic resource management should be combined with other techniques, in particular for dealing with situations when it is not possible to allocate additional resources to fulfill the agreed QoS.

6. SERVICE DEGRADATION

Service degradation avoids refusing clients as a response to overload (as admission control does) by reducing the level of service offered to them under overload conditions, as shown in Figure 6. In general, service degradation can be thought as a special case of service differentiation, where a given customer's class is provided with a degraded level of service when the server is overloaded. In addition, rejecting a request through admission control can be seen as the lowest quality setting for a degraded service.

The initial approach considered in previous works for implementing service degradation was content adaptation [63,64]. These works propose to reduce the quality of static web content (e.g. lower resolution images) provided to clients during overload periods, consuming in this way less system resources. In particular, content adaptation aims to reduce the network bandwidth consumption on the server.

In particular, Abdelzaher and Bhatti [63] evaluate the potential for content adaptation on the web by considering three techniques: image degradation by lossy compression, reduction of embedded objects per page, and reduction in local links. In addition, they present a solution to the overload problem in web servers, which relies on adapting the delivered content. This solution is based on an adaptation agent that monitors the web server load and triggers web content adaptation when server response time increases beyond a pre-computed threshold. Adaptation is evaluated on an Apache web server using only HTTP 1.0 connections.

Chandra *et al.* [64] propose using informed transcoding techniques to provide differentiated service and to dynamically allocate available bandwidth among different client classes, while delivering a high degree of information content (quality factor) for all clients. The authors have modified the Apache web server in order to incorporate transcoding by serving proportionately lower quality variations of images if the average consumed bandwidth for the past half hour is more than the target bandwidth. The server is evaluated using the http_load test client.

One disadvantage of content adaptation is that it is not applicable to many services due to their design. For example, an e-mail or chat service cannot practically degrade service in response to overload: 'lower-quality' e-mail and chat have no meaning. In fact, service degradation as described is not directly applicable to web services. In these cases, another of the described mechanisms must be used.

Table III. Characteristics of works including resource management in their solution.

work	allocations driven by	model	joint?	workload
[39]	monitoring agents issuing SLA violations whenever thresholds are exceeded	dedicated	no	—
[41]	control loop based on thresholds	dedicated	no	RUBiS (dynamic) on Tomcat
[42]	open queuing model	dedicated	no	trace simulation (based on YacSim) of large-scale e-commerce & search-engine traces (dynamic)
[44]	resource usage of containers	shared (S)	no	1 kb static file & CGI script on thttpd
[45]	feedback control loop	shared (S)	no	1 kb to 90 kb static files on Apache
[46]	linear programming formulation	shared (C)	no	static files & CGI scripts on Apache
[47]	open queuing model	shared (C)	no	simulation of World Cup Soccer'98 logs (static)
[48]	resource shares & QoS goals relation computed from runtime measurements	shared (C)	no	static SSL & CGI script on Apache
[49]	predictions on service performance based on runtime measurements	shared (C)	no	Surge (static) on Apache
[50]	applications resource requests based on runtime measurements	shared (C)	no	Traces from Duke Computer Science server on Sun's GridEngine
[6]	predictions from online measurements	shared (C)	yes	Ask Jeeves search (dynamic) on Neptune
[51]	kernel-based online profiling of the resource usage on a dedicated node	shared (C)	no	SpecWeb99 (static & dynamic) on Apache
[52]	applications resource requests and past usage	shared (C)	no	commercial data center & research workgroup environment
[24]	multiclass open queuing model	shared (C)	yes	Internet search (WebGlimpse) & online auction (dynamic)
[55]	runtime measurements of I/O virtualization	shared (V)	no	TPC-W-NYU (dynamic) on JBoss & Streaming media server
[56]	marketplace for computing resources	shared (V)	no	simulation with Apache JMeter
[58]	two-level controller based on fuzzy modeling	shared (V)	no	Java Pet Store (dynamic) & World Cup Soccer'98 logs (static)
[59]	feedback control loop	shared (C)	no	e-commerce (dynamic)
[60]	feedback control loop	dedicated	no	Surge (static) on Apache

Table III. *Continued.*

work	allocations driven by	model	joint?	workload
[57]	feedback control loop	shared (V)	no	RUBiS/TPC-W (dynamic) on Apache
[54]	open queuing model	shared (V)	no	algorithm execution time
[53]	open queuing model	shared (V)	no	discrete event simulation of synthetic workload
[40]	multiclass queuing model	dedicated	no	simulation of transactional & batch workloads
[61]	closed queuing model	shared (C)	no	Surge (static) on Dash
[62]	multiclass open queuing model	dedicated	no	simulation of e-commerce (dynamic)
[43]	open queuing model	dedicated	no	RUBiS/Rubbos (dynamic) on Apache/Tomcat
[34]	applications resource requests based on runtime measurements	shared (C)	yes	RUBiS (dynamic) on Tomcat
[35]	multiclass open queuing model in conjunction with online measurements	dedicated	yes	1kb to 256kb static files & PHP scripts on Apache
[15]	feedback control loop in conjunction with closed queuing model	shared (C)	yes	Trade 6 benchmark (dynamic) on IBM WebSphere

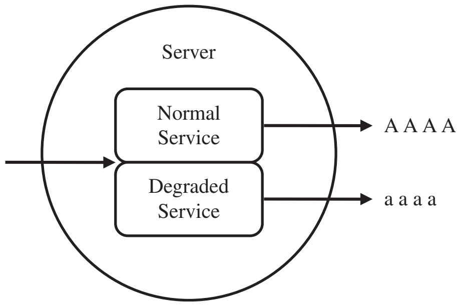

Figure 6. Service degradation technique.

Table IV. Characteristics of works including service degradation in their solution.

work	triggered by	actuation	joint?	workload
[63]	threshold in response time	degrade delivered content (images, embedded objects, local links)	no	static content on Apache
[64]	threshold in avg consumed bandwidth	deliver lower quality images	no	static content on Apache
[65]	threshold in resource utilization	degrade delivered content within SLA limits	yes	64kb static images on Apache
[35]	threshold in request drop rate	degrade performance within SLA limits	yes	1–256kb static files & PHP scripts on Apache

For this reason, recent works [35,65] have reinterpreted the concept of service degradation. For instance, Urgaonkar and Shenoy [35] implement what they call 'QoS adaptive degradation'. This approach considers that during overload conditions (when the request drop rate exceeds a certain predefined value), the performance of admitted requests can be degraded within the limits established by the SLA. The same concept is applied in [65], but named 'QoS adaptation'. In this work, the authors use the mechanisms for content adaptation described in [63] for providing degraded service within the values indicated in the SLA when the resource utilization exceeds a predefined threshold.

Other works do not directly implement service degradation mechanisms, but rather signal overload to applications in a way that allows them to degrade if possible. For example, in SEDA [26], stages can obtain the current 90th-percentile response time measurement as well as enable or disable the stage's admission control mechanism. This allows a service to implement the degradation by periodically sampling the current response time and comparing it to the target. If service degradation is ineffective, because the load is too high to support even the lowest quality setting, the stage can re-enable admission control to cause requests to be rejected.

Table IV summarizes the main characteristics (concerning this technique) of the works using service degradation in their solution. This includes how the service degradation mechanism is triggered, the actuation performed, whether this work uses also other techniques, and its target workload.

Service degradation can be thought of as a complementary technique that can be used to enhance a given performance management solution. Using it alone can contribute to delay overload, but it needs to be combined with other techniques to be fully effective. In addition, it is mainly focused on static web content, though it could be reinterpreted to fit also in web services environments.

7. CONTROL THEORETICAL APPROACHES

Control theory has been widely used to adjust the behavior of traditional dynamical systems. Internet systems also fit in the definition of a dynamical system, and for this reason, a great number of works [15,30,31,45,57,59,60,65] have proposed the use of control theory for performance management in the context of Internet applications. As a rule of thumb, these works propose the use of a closed-loop controller that controls the parameters of the actuation technique (generally admission control and/or resource provisioning) using feedback information from the system. A typical architecture of such a system is shown in Figure 7. Notice that in a system managed by a closed-loop controller (a.k.a. feedback controller), the output of the system y is fed back to the reference value r, through a sensor measurement. The controller then takes the error e (i.e. the difference) between the reference and the output to change the inputs u to the system under control (the server) by means of an actuator. As commented, admission control and dynamic resource provisioning have been typically used as actuators in the related literature.

In particular, Abdelzaher *et al.* [65] describe the use of classical feedback control theory for performance control of a web server, achieving in this way overload protection, performance guarantees, and service differentiation in the presence of load unpredictability. The proposed architecture extends pure admission control schemes with the degradation of clients QoS, which is accomplished using content adaptation. The authors design a utilization control loop that regulates the extent of degradation to satisfy a pre-specified utilization bound. The reference resource demands are derived from served request rate and delivered byte bandwidth using a time-varying linear model of a static web server (i.e. Apache web server). The workload for the experiments is generated using the httperf tool.

Blanquer *et al.* [30] describe Quorum, a non-invasive software approach to QoS provisioning for large-scale Internet services which ensures reliable QoS guarantees using traffic shaping and admission control at the entrance of the site, and response monitoring at the exit. Quorum treats the site as an unknown ‘black-box’ system, intercepts the request and response streams entering and leaving the site, and uses feedback-driven techniques (based on weighted fair queuing) to

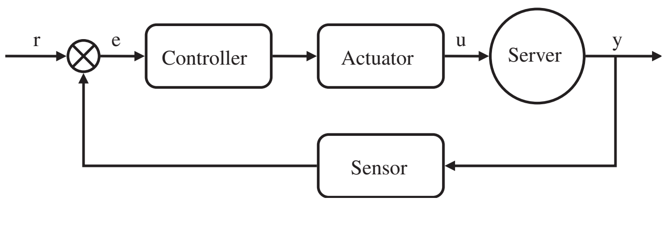

Figure 7. Typical architecture of a control theoretical approach.

dynamically control the proportions in which requests of different classes must be forwarded to the server. In this system, a request will be dropped when the sum of the time it has been queued and its observed compute time is higher than its maximum allowed response time. The evaluation, based on the Teoma Internet Search Engine and the TPC-W benchmark, includes a comparison with the Neptune system [6].

Diao *et al.* [31] describe a feedback control mechanism based on a ‘black-box’ model that meets given CPU and memory utilization targets by tuning Apache server parameters (the number of server processes (MaxClients) and the per-connection idle timeout (KeepAlive Timeout)). However, these parameters do not directly address metrics of interest to the web site, such as response time or throughput. In addition, reducing the number of server processes leads to increased likelihood of stalling incoming connections. Although this can effectively protect server from overload, it results in the poor client perceived performance. The authors also limit their attention to static web content. This proposal is evaluated on the Apache web server using the httperf tool. The generated workload is extracted from the WebStone benchmark.

Karlsson *et al.* [59] propose a non-intrusive approach that interposes a fair-queuing scheduler on the request path between a service and its clients to intercept incoming requests and enforce proportional sharing of the service resources among workloads. The authors have designed an adaptive optimal MIMO controller that dynamically sets the share of each workload based on their observed throughputs and response times. The experimental evaluation is conducted using two different systems: a 3-tier e-commerce system that mimics real-world users behavior and receives requests from clients running the httperf tool, and an NFS file server that is shared by multiple workloads.

Lu *et al.* [60] introduce a feedback control architecture for adapting resource allocation such that differentiated connection delays between different service classes in web servers under HTTP 1.1 are achieved. Using the Root Locus method, the authors design a feedback controller that computes the process proportions for each class to maintain the desired connection delays (the time interval between the arrival of a TCP connection request and the time, the connection is accepted by a server process). The controller assumes linear behavior of the system. A modified Apache web server is used for implementing the adaptive architecture and, for this reason, the evaluation only considers static web content (in the form of the SURGE benchmark).

Finally, Padala *et al.* [57] address the problem of dynamically controlling resource allocations to individual components of complex, multi-tier enterprise applications in a shared hosting environment. The authors propose a two-layered controller architecture: a utilization controller that controls the resource allocation for a single application tier and an arbiter controller that controls the resource allocations across multiple application tiers and multiple application stacks sharing the same infrastructure. This controller dynamically adjusts the CPU shares to each VM in order to meet application-level QoS goals while achieving high resource utilization in the data center. In order to accomplish this, the controller sets CPU shares that lead to a 80% utilization level for each VM. This proposal is evaluated in a virtualized platform (with Xen Hypervisor), using two two-tier applications: the RUBiS auction site benchmark and a Java implementation of TPC-W, both deployed in the Apache web server.

Table V summarizes the main characteristics (concerning this technique) of the works using control theory in their solution. This includes the reference signal of the system, the actuation performed, whether this work also uses other techniques, and its target workload.

Table V. Summary of works including control theory in their solution.

work	reference	actuation	joint?	workload
[45]	response time	set CPU entitlement	no	1–90 kb static files on Apache
[65]	resource utilization	degradation of clients QoS	yes	64kb static images on Apache
[30]	response time & throughput	adjust fair queuing weights	no	TPC-W & Teoma Internet Search Engine (dynamic) on Apache/Tomcat/Neptune
[31]	CPU & memory	adjust MaxClients & KeepAlive Timeout	no	WebStone (static) on Apache
[32]	response time	adjust request acceptance probability	yes	TPC-W (dynamic) on Tomcat
[33]	response time	adjust request acceptance probability	yes	TPC-W (dynamic) on Tomcat
[57]	VM resource utilization	set CPU share allocated to each VM	no	RUBiS/TPC-W (dynamic) on Apache
[59]	response time & throughput	set resource share for each workload	no	e-commerce (dynamic)
[60]	avg connection delay	allot server processes to each class	no	Surge (static) on Apache
[15]	response time	set the amount of resources to allocate to each service class	yes	Trade 6 benchmark (dynamic) on IBM WebSphere

Although control theory allows building systems able to adapt the variable environment conditions, classical solutions need that these systems are trained before at different operating points to determine the control parameters for a given workload, which have to be re-computed when the workload characteristics change. In addition, they typically assume a linear relationship between the control parameters and the QoS goals. Though this can be enough for controlling some QoS goals (e.g. throughput), other goals, such as response time, cannot be accurately captured with linear models.

In order to overcome these problems, the system should be able to estimate repeatedly the dynamic model based on online input–output measurements, and compute the controller parameters online based on the current estimated model. This technique, which is known as adaptive control theory, is used in some works [32,33,45,59] to control the performance of the system. This helps in adjusting the model to the workload dynamics to some extent, though it remains intrinsically linear. For this reason, the adaptive controllers usually respond slowly to sudden workload changes. Similarly, Li *et al.* [66] make use of system identification techniques to model at real-time the flow in the stages of the SEDA system and to optimize the controlled parameter configurations at runtime.

8. QUEUING MODEL-BASED APPROACHES

Similar to control theory, queuing theory has also been broadly used for modeling the behavior of complex systems. As a particular case, a large number of works [15,24,40,43,47,53,54,61,62] propose queuing model-based models to achieve the performance guarantees in Internet systems. Their basic idea is to model the system behavior using a queuing model and use the classical results

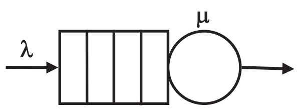

Figure 8. System-level model of a web system.

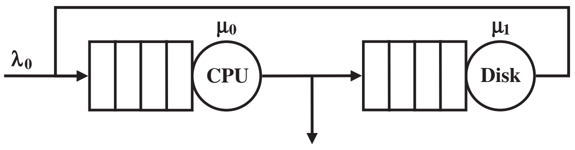

Figure 9. Component-level model of a web system (resource level).

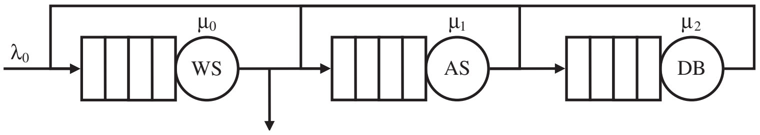

Figure 10. Component-level model of a web system (tier level).

from queuing theory to predict the control values (e.g. admitting probability, resource allotments) necessary to achieve the specified delay targets given the current observed workload.

Queuing models for Internet applications can be classified in system-level and component-level models. System-level models (a.k.a. Single Queue models) view the system as a 'black-box', as shown in Figure 8, where λ represents the arrival distribution of incoming connections and μ the service time distribution. Component-level models (a.k.a. Queuing Networks models) use a more detailed level by considering explicitly the different system resources (e.g. processors, disks, etc.) or tiers (web, application, and database tiers), as shown for instance in Figures 9 and 10, respectively. Notice that in component-level models, each resource is modeled by a queue and the various queues that represent a system are interconnected.

In addition, one can also distinguish between single-class and multi-class models. In single-class models, all requests have similar characteristics (i.e. they belong to the same class), and thereby they receive the same service time in the resources. In multi-class models, several classes of requests are considered. These models can reflect situations where different customer categories have different frequency of arrival and/or different resource service demands. The models presented so far are single-class models, while the model shown in Figure 11 considers three different classes, each one with different arrival rates (λ_0, λ_1, λ_2) and service distributions (ϕ_0, ϕ_1, ϕ_2).

Finally, one can also distinguish between open and closed models. Open models do not place any limits on the number of requests present in the system, while closed models have a fixed number of requests per class. The models presented so far are open models, while the model shown in Figure 12 has a fixed number of clients that issue another request to the system after some 'think-time' (Z station).

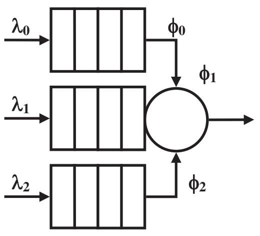

Figure 11. Multi-class component-level model of a web system.

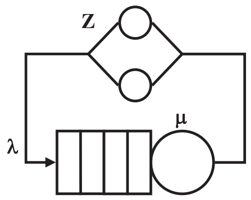

Figure 12. Closed component-level model of a web system.

Queuing models are usually referred using Kendall Notation, which details (separated using slashes) the arrival process (e.g. M if the interarrival times are exponentially distributed, G if general), the service time distribution, the number of servers, the system capacity (i.e. those waiting for service and those receiving), the population size (i.e. the total number of potential users), and the service discipline (i.e. the order of arrival).

With respect to the specific proposals for using queuing models for the performance management of Internet applications, Abrahao *et al.* [53] consider the problem of hosting multiple third-party Internet services in a cost-effective manner so as to maximize a provider's business objective. For this purpose, the authors present a dynamic capacity management framework based on an optimization model, which links a cost model based on SLA contracts with two analytical open queuing-based performance models, namely M/M/1 and M/G/1 Processor Sharing, in an attempt to adapt the platform to changing capacity needs in real time. The authors consider three approximations to compute the tail probability distribution of the response time, namely Markov, Chebyshev, and Percentile. In the same way as [54], this work also considers virtualized execution environments. Evaluation results are derived from the discrete event simulation of the environment considered using the nonlinear solver DONLP2.

Almeida *et al.* [54] use queuing models to address the resource management problem in autonomic service-oriented architectures. A key feature of the environment under consideration in this paper is its virtualization scheme, which partitions the physical resources into multiple virtual ones, creating isolated virtual machines, each running at a fraction of the total (physical) system capacity and dedicated to serve a single Web Service class. The authors first identify two key problems that must be solved by service providers, namely, the short-term resource allocation problem and the long-term capacity planning problem. Then, for the short-term problem, they formalize it by means of a nonlinear optimization model, resolved via a fixed point iteration (FPI) technique, which identifies the optimal resource allocation across a set of service invocations by maximizing a provider's revenues, while satisfying customers' QoS constraints and minimizing the cost of resources utilization. Predictions of the performance metrics used for the optimization are obtained through an analytical M/G/1 open queuing model for each VM. The evaluation consists only of running experiments varying the values of the model parameters and evaluating the algorithm execution time.

Bennani and Menasce [40] address resource allocation problems in autonomic data centers. The authors combine the use of analytic predictive multi-class queuing network models and combinatorial search techniques to design a controller for determining the required number of servers for each application environment in order to continuously meet the SLA goals under a dynamically varying workload. Response time of transactional workloads is estimated using the multi-class open queuing network models presented in [67], while response time of batch workloads is estimated using the multi-class closed queuing network models described also in [67]. The evaluation is conducted through simulation experiments, considering both transactional and batch workloads.

Chandra *et al.* [47] present techniques for the dynamic resource allocation in shared web servers. The main goal of these techniques is to react to transient system overloads by incorporating online system measurements. The authors model a server resource with multiple class-specific queues with GPS for sharing the resource capacity among the queues, and then employ a nonlinear optimization technique (i.e. Lagrange multiplier method) driven by the transient queue behavior in this model to derive the resource allocations. The authors propose also a prediction algorithm that estimates the workload parameters of applications in the near future using online measurements. This proposal is evaluated using a simulation study based on both synthetic workloads and real-world web traces. These traces are extracted from the World Cup Soccer'98 server logs, which considers only static web content.

Doyle *et al.* [61] present a model-based resource provisioning scheme that performs resource allocation based on the performance modeling of a shared server cluster, focusing on coordinated management of memory and storage. The authors incorporate internal models for capturing the service behavior and predict the value of resource allocations under changing load. In particular, the predicted hit ratio for the memory is estimated using Zipf-distribution, while the average response time from storage is estimated using a simple closed queuing model of CPU and storage resources for each storage server. This work builds on Muse [49] and uses web traces and synthetic loads, both based only in static web content, to conduct the evaluation. Requests are generated using the Surge tool.

Liu *et al.* [62] propose an analytic approach for provisioning of resources toward the maximization of SLA profits. The authors model the server farm using a GPS closed queuing model with a multi-class queue for each server, and formulate the SLA profits maximization problem as a network flow model with a separable set of concave objective functions at the servers (or sink nodes) that are based on queuing-theoretical formulas derived to capture the system dynamics. Then, the authors solve this problem via a FPI (nonlinear optimization driven by steady-state queue behavior). The benefits of the proposed approach are investigated through simulation.

Urgaonkar *et al.* [43] describe a dynamic capacity provisioning technique for multi-tier Internet applications running in dedicated hosting platforms, so that they can service their peak workload demand while meeting contracted response time guarantees. The authors use a G/G/1-based open queuing model to determine how much resources to allocate to each tier of the application based on the estimated workload and a combination of predictive and reactive methods that determine when to provision these resources, both at large and small time scales. The evaluation is conducted through two open-source multi-tier applications, namely RUBiS, which implements the core functionality of an eBay like auction site, and Rubbos, which is a bulletin-board application modeled after an online news forum like Slashdot. The 3 tiers are implemented using Apache, Tomcat, and MySQL.

Finally, Zhu *et al.* [24] present a service differentiation scheme, called demand-driven service differentiation, for cluster-based network services. This scheme includes a flexible mechanism for defining service quality classes (each class with an assigned priority) and a resource scheduling algorithm that dynamically adjusts server partitions based on resource demands, ensuring higher-priority classes get better services while lower-priority classes are not over-sacrificed when there are enough resources available. This is accomplished by continuously monitoring the system status and periodically adjusting resource allocations using a multi-class open M/M/N queuing network that approximates the server behavior as a guide. This scheme also uses priority-based admission control to drop excessive requests when the upper bounds in stretch factor (response time/service demand) are exceeded. This approach is limited in its ability to accommodate a large number of service

Table VI. Summary of works including queuing models in their solution.

work	queuing model	mult?	open/closed	joint?	workload
[42]	G/G/N model with FCFS scheduling on each server	no	open	no	trace simulation (based on YacSim) of large-scale e-commerce & search-engine traces (dynamic)
[53]	M/M/1 and M/G/1 with processor sharing	no	open	no	discrete event simulation of synthetic workload
[54]	M/G/1 for each VM	no	open	no	algorithm execution time
[40]	multi-class models to estimate response time	yes	both	no	simulation of transactional & batch workloads
[47]	GPS for each resource	yes	open	no	simulation of World Cup Soccer'98 logs (static)
[61]	simple model considering CPU and storage	no	closed	no	Surge (static) on Dash
[62]	GPS with a multi-class queue for each server	yes	closed	no	simulation of e-commerce (dynamic)
[43]	G/G/1 model for each server at each tier	no	open	no	RUBiS/Rubbos (dynamic) on Apache/Tomcat
[32]	web application as a M/GI/1/PS queue	no	closed	yes	TPC-W (dynamic) on Tomcat
[33]	web application as a M/GI/1/PS queue	no	closed	yes	TPC-W (dynamic) on Tomcat
[35]	G/G/1 model for each single replica of a service	yes	open	yes	1–256 kb static files & PHP scripts on Apache
[24]	one-tier architecture as a M/M/N queue	yes	open	yes	WebGlimpse (Internet search) & online auction (dynamic)
[15]	queue with FCFS scheduling for each class	yes	closed	yes	Trade 6 benchmark (dynamic) on IBM WebSphere

classes. The experiments in this paper involve two applications: Internet search (i.e. WebGlimpse) and online auction.

Table VI summarizes the main characteristics (concerning this technique) of the works using queuing models in their solution. This includes the parameters of the model, whether the model is multi-class, whether it is open, closed or both, whether this work also uses other techniques, and its target workload.

Although queuing models are useful for steady-state analysis, they do not handle transients accurately. In addition, they are difficult to solve analytically for complex arrival patterns and service time distributions. This is not negligible, since choosing simple approximations of arrival and service time distributions can lead to incorrect the choice of parameters. For these reasons, performance management using exclusively queuing models is not adequate.

9. CONTROL THEORETICAL AND QUEUING MODEL-BASED APPROACHES

Given the benefits and the limitations of control theory and queuing models for achieving performance guarantees in Internet servers, some authors [32,33] have approached this problem by combining both techniques. In these works, the basic control parameters of the system are predicted using a queuing model. At this point, control theory is used to correct the residual error incurred by using this prediction.

In particular, Kamra *et al.* [32] present Yaksha, a control-theoretical approach for admission control in multi-tier e-commerce applications that both prevent overload and enforce absolute client response times. Yaksha implements a self-tuning PI controller for admission control of client HTTP requests. The controller uses a closed M/GI/1 Processor Sharing queuing model for representing a multi-tier e-commerce application. In this way, no parameter setting is required beyond a target response time, which is enforced by adjusting the acceptance probability of requests to the system. The authors have implemented their approach as a proxy, which operates by taking simple external measurements of the client response times. The evaluation is conducted using the TPC-W benchmark deployed on a Tomcat server.

Liu *et al.* [33] propose a scheme for the autonomous performance control of web applications called queuing-model-based adaptive control. This approach combines both the modeling power of queuing theory and self-tuning power of adaptive control. Based on the measurements from the server to be controlled, the authors use a closed M/GI/1 Processor Sharing queuing model predictor to estimate the admitting probability necessary to achieve the specified average delay target given the currently observed average workload. Then, an online adaptive feedback loop enforces admission control of the incoming requests to ensure that the desired response time target is met in order to correct possible errors due to the inaccuracies in the queuing model. The evaluation of this approach uses the TPC-W benchmark deployed on a Tomcat server.

10. OBSERVATION-BASED APPROACHES USING RUNTIME MEASUREMENTS

As commented in Section 2, all the proposals use runtime measurements representing the dynamics of the system in some way to make the required computations needed to configure the parameters of the technique they apply. We consider that a work is observation-based when these runtime measurements are not used to feed a control theory or queuing model, but to make some other *ad hoc* computations. According to this, all the works instead those described in Sections 7, 8, and 9 will fall in this category, though one of the most representative works is the one by Chandra *et al.* [48].

This work describes an observation-based approach for self-managing web servers on shared architectures. The main goal is that servers can adapt to changing workloads while maintaining the QoS requirements of different classes. In this approach, the adaptation framework monitors the system state continuously and adjusts the resource allocations for each class (primarily the accept queue and the CPU) based on the monitored performance and the desired QoS goal. A key feature of this technique is that it can handle multiple OS resources in tandem. The evaluation is done over a modified Apache web server using static workloads with SSL, although they also examine a synthetic CGI script that blocks for a random amount of time. The workload generator used at the clients is httperf. Observation-based approaches such as this are suited for handling varying workloads and nonlinear behaviors.

11. OVERALL APPROACHES COMBINING SEVERAL TECHNIQUES

Some latter works [15,34,35] have demonstrated that the most effective way to prevent Internet applications overload and provide performance guarantees require the combination of several techniques, instead of considering each one in isolation. In these works, the different techniques compensate the limitations of the others.

In particular, Guitart *et al.* [34] combine several techniques such as admission control based on SSL connections differentiation and dynamic resource provisioning for overload prevention of e-commerce applications and efficient utilization of platform resources. Their approach is based on the cooperation between a global resource manager and the applications for managing the resources. The resource manager distributes periodically the available processors among web applications using e-business indicators (i.e. customer's priority) as well as traditional indicators (i.e. response time). The applications adapt their behavior to the allocated processors by limiting the accepted load using an admission control mechanism (proposed by the authors in their previous work [25]) in order to avoid QoS degradation. In order to evaluate their approach, the authors use the httperf generator to issue secure requests to a Tomcat server with the RUBiS benchmark deployed.

Pacifici *et al.* [15] present the architecture of a performance management system for managing the response time of different multi-tiered web applications deployed on clustered web servers. The authors introduce an additional tier, the proxy tier, which classifies incoming requests into request queues according to the configured policy. The proxy tier has a scheduler for each gateway (defined as a set of queues (one for each traffic class) corresponding to a given application) that selectively queues and releases requests (using a weighted round robin scheme) to manage response time goals and protect computing resources from overload. The platform has a resource controller that monitors the request response times and other performance metrics, and periodically re-computes the parameter values for the resource allocation that the schedulers in the proxy use. The authors formulate the resource allocation problem as an optimization problem with an objective function derived from the class utility functions and resolve it using a feedback control loop and an independent closed queuing network model with FCFS scheduling for each traffic class. The system also computes resource request signals that may be used to drive a dynamic placement management system, as done by [68]. Experimental results are based on two different deployments (one configured to use direct JDBC connections to access the database and the other uses a layer of Enterprise JavaBeans) of the Trade6 benchmark from IBM (which models an online stock brokerage application) in the WebSphere server.

Finally, Urgaonkar and Shenoy [35] describe Cataclysm, a system for handling extreme overloads in a dedicated hosting platform running multiple Internet services. Cataclysm performs overload control bringing together admission control, adaptive QoS degradation, and dynamic provisioning of platform resources into an integrated system, demonstrating that the most effective way to handle overload must consider the synergistic combination of techniques. The authors propose a low overhead admission control mechanism that preferentially admits important requests by performing the admission test on queued requests in the decreasing order of importance. With this test, a request is admitted if and only if the sum of the estimated queuing delay seen by a request and its actual service time does not exceed the desired response time. The proposed dynamic provisioning mechanism employs a multi-class open queuing model of a replicable application in conjunction with online measurements to dynamically vary the number of servers allocated to each application. In the proposed model, each single replica of a service is modeled as a $G/G/1$ queuing system. For evaluating their approach, the authors use the httperf tool for generating requests to Apache servers with PHP support enabled.

As a summary of this paper, Table VII shows the techniques that each introduced work uses. In this table, RS refers to Request Scheduling, SD to Service Differentiation, AC to Admission Control, RM to Resource Management, SG to Service Degradation, CT to Control Theory, and QM to Queuing Model. A shaded cell indicates that the work uses that technique.

12. DISCUSSION

Since the appearance of Internet applications, they have evolved immensely. The main changes include going from simple static web content (HTML pages and images) to dynamic and/or secure content, going from simple one-tier architectures to multi-tier ones, or going from a single server attending client's requests to server farms. Notice that all these changes have resulted in better service for the clients of these kinds of applications (e.g. sophisticated contents, lower response time, higher availability, etc.) but have also involved a higher degree of complexity in their management.

As shown in this paper, this evolution has conditioned the proposals for managing the performance of Internet applications. When considering the actuation performed to manage the performance, the former proposals tackled the problem using a single technique, demonstrating that this could be effective in some measure for specific scenarios. In particular, request scheduling can improve the response time of higher-priority requests, but this technique is not enough on overload situations. In the same way, allocating additional capacity to the application is also a very effective solution until all the provider's resources get allocated. The same applies to the degradation of the service performance, which can also be carried out while the agreed limit in the SLA is not exceeded. Finally, turning away the excess requests that cannot be attended can serve to avoid the degradation

Table VII. Summary of works according to used techniques.

work	RS	SD	AC	RM	SG	CT	QM
[1]	▨	▨					
[2]	▨	▨					
[3]	▨	▨					
[4]	▨	▨					
[5]	▨	▨					
[6]	▨	▨		▨			
[7]	▨	▨					
[8]	▨	▨					
[9]	▨	▨					
[10]	▨	▨					
[11]	▨	▨	▨				
[12]	▨	▨	▨				
[13]	▨	▨	▨				
[14]	▨	▨	▨				
[15]	▨	▨		▨		▨	▨
[16]	▨	▨	▨				
[17]	▨	▨	▨				
[19]		▨	▨				
[18]			▨				
[20]		▨	▨				
[21]		▨	▨				
[22]		▨	▨				
[23]		▨	▨				
[24]		▨	▨	▨			
[25]		▨	▨				
[26]			▨				
[28]		▨	▨				
[29]		▨	▨				
[30]		▨	▨			▨	
[31]			▨			▨	
[32]			▨			▨	▨
[33]			▨			▨	▨
[34]		▨	▨	▨			
[35]		▨	▨	▨	▨		▨

Table VII. *Continued.*

work	RS	SD	AC	RM	SG	CT	QM
[39]		▨		▨			
[40]		▨		▨			▨
[41]		▨		▨			
[42]		▨		▨			▨
[43]		▨		▨			▨
[44]		▨		▨			
[45]		▨		▨		▨	
[46]		▨		▨			
[47]		▨		▨			▨
[48]		▨		▨			
[49]		▨		▨			
[50]		▨		▨			
[51]		▨		▨			
[52]		▨		▨			
[53]		▨		▨			▨
[54]		▨		▨			▨
[55]		▨		▨			
[56]		▨		▨			
[57]		▨		▨		▨	
[58]		▨		▨			
[59]		▨		▨		▨	
[60]		▨		▨		▨	
[61]		▨		▨			▨
[62]		▨		▨			▨
[63]					▨		
[64]		▨			▨		
[65]		▨			▨	▨	

of the server performance, though under extreme overloads a noticeable amount of requests could be rejected, which is inconvenient from the business perspective.

According to this, the latter works have shown that the combination of different approaches can lead to better solutions, which exploit the advantages of each technique while compensating their drawbacks. In addition, as Internet applications have become commonplace in e-commerce transactions, in an effort to accommodate their solutions to the new requirements of Internet applications, recent works have incorporated the business value to their performance management decision engine. Similarly, due to the progressive increase in Internet applications, complexity, performance solutions have also evolved to be self-adaptive and autonomic, in order to minimize the external intervention needed for configuring and maintaining the system.

The same occurs when considering the mechanism used to take the performance management decisions. Each individual technique is intended for a specific scenario. In particular, control theory allows building systems that are able to react to variations in the environment conditions. However, control parameters must be re-computed when the workload characteristics change, though later proposals allow for the adaptive optimization of the controller parameters at runtime based on dynamically estimated models. In addition, classical control theoretical approaches assume a linear relationship between the control parameters and the QoS goals, which is inadequate for some QoS metrics. Queuing models can provide an accurate analysis of the steady state, and for this reason, they are useful for building predictive systems. However, they do not handle transients accurately, and this limits their use for taking reactive decisions. In addition, they are difficult to solve analytically for complex arrival patterns and service time distributions. Finally, observation-based approaches can be a good option for building reactive systems when handling varying workloads and nonlinear behaviors. According to this, it seems reasonable that the latter works combine some of these approaches in their solutions.

13. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this paper, we have introduced a complete taxonomy that characterizes and classifies the different approaches suggested in the literature for managing the performance of Internet applications. The taxonomy considers two different perspectives. On one hand, techniques are grouped depending on the actuation performed to manage the performance. This includes request scheduling, admission control, service differentiation, service degradation, dynamic resource management. On the other hand, techniques are also grouped depending on the mechanism used to take the performance management decisions. This includes control theory, queuing models, and observation-based approaches using runtime measurements. Then, we have conducted a survey on different related works, mapping them into the various taxonomy categories.

Future proposals in this area will have to deal with the constant evolution of Internet applications, and the requirements of the emerging models for Internet-based computation (e.g. cloud computing, service-oriented computing). After surveying the related work in this area, it seems that these future proposals could follow the basic principles introduced by the latter works, that is, the synergistic combination of several techniques, the use of high-level decision policies including business metrics, and the use of adaptive engines to dynamically accommodate the system behavior to environmental changes.

ACKNOWLEDGEMENTS

This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625 and by the BSC (Barcelona Supercomputing Center).

REFERENCES

1. Crovella M, Frangioso R, Harchol-Balter M. Connection scheduling in web servers. *Second Symposium on Internet Technologies and Systems* (*USITS'99*), Boulder, CO, U.S.A., 11–14 October 1999.

2. Harchol-Balter M, Schroeder B, Bansal N, Agrawal M. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems* (*TOCS*) 2003; **21**(2):207–233.
3. Schroeder B, Harchol-Balter M. Web servers under overload: How scheduling can help. *ACM Transactions on Internet Technology* 2006; **6**(1):20–52.
4. Rawat M, Kshemkayani A. SWIFT: Scheduling in web servers for fast response time. *Second IEEE International Symposium on Network Computing and Applications* (*NCA 2003*), Cambridge, MA, U.S.A., 16–18 April 2003; 51–58.
5. Zhou J, Zhang C, Yang T, Chu L. Request-aware scheduling for busy internet services. *Twenty-Fifth IEEE INFOCOM 2006*, Barcelona, Spain, 23–29 April 2006.
6. Shen K, Tang H, Yang T, Chu L. Integrated resource management for cluster-based internet services. *Fifth Symposium on Operating Systems Design and Implementation* (*OSDI'02*), Boston, MA, U.S.A., 9–11 December 2002.
7. Almeida J, Dabu M, Manikutty A, Cao P. Providing differentiated quality-of-service in web hosting services. *Workshop on Internet Server Performance* (*WISP'98*)*(in conjunction with SIGMETRICS'98)*, Madison, WI, U.S.A., 23 June 1998.
8. Alonso J, Guitart J, Torres J. Differentiated quality of service for e-commerce applications through connection scheduling based on system-level thread priorities. *Fifteenth Euromicro Conference on Parallel*, *Distributed and Network-based Processing* (*PDP'07*), Naples, Italy, 7–9 February 2007.
9. Menasce D, Almeida V, Fonseca R, Mendes M. Business-oriented resource management policies for e-commerce servers. *Performance Evaluation* 2000; **42**(2–3):223–239.
10. Totok A, Karamcheti V. Improving performance of internet services through reward-driven request prioritization. *Fourteenth International Workshop on Quality of Service* (*IWQoS*), New Haven, CT, U.S.A., 19–21 June 2006; 60–71.
11. Bhatti N, Friedrich R. Web server support for tiered services. *IEEE Network* 1999; **13**(5):64–71.
12. Carlstrom J, Rom R. Application-aware admission control and scheduling in web servers. *Twenty-first IEEE INFOCOM 2002*, New York, U.S.A., 23–27 June 2002; 506–515.
13. Chen H, Mohapatra P. Overload control in QoS-aware web servers. *Computer Networks* 2003; **42**(1):119–133.
14. Elnikety S, Nahum E, Tracey J, Zwaenepoel WA. Method for transparent admission control and request scheduling in e-commerce web sites. *Thirteenth International Conference on World Wide Web* (*WWW'04*), New York, U.S.A., 17–22 May 2004; 276–286.
15. Pacifici G, Segmuller W, Spreitzer M, Steinder M, Tantawi A, Youssef A. Managing the response time for multi-tiered web applications. *Technical Report RC23651*, IBM, 2005.
16. Verma A, Ghosal S. On admission control for profit maximization of networked service providers. *Twelfth International World Wide Web Conference* (*WWW'03*), Budapest, Hungary, 20–24 May 2003; 128–137.
17. Voigt T, Tewari R, Freimuth D, Mehra A. Kernel mechanisms for service differentiation in overloaded web servers. *USENIX Annual Technical Conference*, Boston, MA, U.S.A., 25–30 June 2001; 189–202.
18. Iyer R, Tewari V, Kant K. Overload control mechanisms for web servers. *Workshop on Performance and QoS of Next Generation Networks*, Nagoya, Japan, 2000; 225–244.
19. Chen X, Chen H, Mohapatra P. ACES: An efficient admission control scheme for QoS-aware web servers. *Computer Communications* 2003; **26**(14):1581–1593.
20. Jamjoom H, Reumann J. QGuard: Protecting internet servers from overload. *Technical Report CSE-TR-427-00*, University of Michigan, 2000.
21. Kanodia V, Knightly EW. Ensuring latency targets in multiclass web servers. *IEEE Transactions on Parallel and Distributed Systems* 2003; **14**(1):84–93.
22. Li K, Jamin S. A measurement-based admission-controlled web server. *Nineteenth IEEE INFOCOM 2000*. Tel Aviv: Israel, 26–30 March 2000; 651–659.
23. Voigt T, Gunningberg P. Adaptive resource-based web server admission control. *Seventh IEEE International Symposium on Computers and Communication* (*ISCC'02*). Taormina/Giardini Naxos: Italy, 1–4 July 2002; 219–224.
24. Zhu H, Tang H, Yang T. Demand-driven service differentiation in cluster-based network servers. *Twentieth IEEE INFOCOM 2001*, Anchorage, Alaska, U.S.A., 22–26 April 2001; 679–688.
25. Guitart J, Carrera D, Beltran V, Torres J, Ayguade E. Designing an overload control strategy for secure e-commerce applications. *Computer Networks* 2007; **51**(15):4492–4510.
26. Welsh M, Culler D. Adaptive overload control for busy internet servers. *Fourth Symposium on Internet Technologies and Systems* (*USITS'03*), Seattle, WA, U.S.A., 26–28 March 2003.
27. Welsh M, Culler D, Brewer E. SEDA: An architecture for well-conditioned, scalable internet services. *Eighteenth Symposium on Operating Systems Principles* (*SOSP'01*), Banff, Canada, 21–24 October 2001; 230–243.
28. Cherkasova L, Phaal P. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers* 2002; **51**(6):669–685.
29. Voigt T, Gunningberg P. Kernel-based control of persistent web server connections. *ACM SIGMETRICS Performance Evaluation Review* 2001; **29**(2):20–25.
30. Blanquer J, Batchelli A, Schauser K, Wolski R. Quorum: Flexible quality of service for internet services. *Second Symposium on Networked Systems Design and Implementation* (*NSDI'05*), Boston, MA, U.S.A., 2–4 May 2005; 159–174.

31. Diao Y, Gandhi N, Hellerstein JL, Parekh S, Tilbury DM. Using MIMO feedback control to enforce policies for interrelated metrics with application to the apache web server. *Eighth IEEE/IFIP Network Operations and Management Symposium* (*NOMS 2002*), Florence, Italy, 15–19 April 2002; 219–234.
32. Kamra A, Misra V, Nahum E. Yaksha: A controller for managing the performance of 3-tiered websites. *Twelfth International Workshop on Quality of Service* (*IWQoS 2004*), Montreal, Canada, 7–9 June 2004.
33. Liu X, Heo J, Sha L, Zhu X. Adaptive control of multi-tiered web applications using queueing predictor. *Tenth IEEE/IFIP Network Operations and Management Symposium* (*NOMS 2006*), Vancouver, Canada, 3–7 April 2006; 106–114.
34. Guitart J, Carrera D, Beltran V, Torres J, Ayguade E. Dynamic CPU provisioning for self-managed secure web applications in SMP hosting platforms. *Computer Networks* 2008; **52**(7):1390–1409.
35. Urgaonkar B, Shenoy P. Cataclysm: Scalable overload policing for internet applications. *Journal of Network and Computer Applications* (*JNCA*) 2008; **31**:891–920.
36. Andrzejak A, Arlitt M, Rolia J. Bounding the resource savings of utility computing models. *Technical Report HPL-2002-339*, HP Labs, 2002.
37. Chandra A, Goyal P, Shenoy P. Quantifying the benefits of resource multiplexing in on-demand data centers. *First Workshop on Algorithms and Architectures for Self-Managing Systems* (*Self-Manage 2003*), San Diego, CA, U.S.A., 11 June 2003.
38. Chandra A, Shenoy P. Effectiveness of dynamic resource allocation for handling internet flash crowds. *Technical Report TR03-37*, Department of Computer Science, University of Massachusetts, U.S.A., 2003.
39. Appleby K, Fakhouri S, Fong L, Goldszmidt G, Krishnakumar S, Pazel D, Pershing J, Rochwerger B. Oceano-SLA-based management of a computing utility. *IFIP/IEEE Symposium on Integrated Network Management* (*IM 2001*), Seattle, WA, U.S.A., 14–18 May 2001; 855–868.
40. Bennani M, Menasce D. Resource allocation for autonomic data centers using analytic performance models. *Second International Conference on Autonomic Computing* (*ICAC'05*), Seattle, WA, U.S.A., 13–16 June 2005; 229–240.
41. Bouchenak S, Palma ND, Hagimont D, Taton C. Autonomic management of clustered applications. *International Conference on Cluster Computing* (*Cluster 2006*), Barcelona, Spain, 25–28 September 2006.
42. Ranjan S, Rolia J, Fu H, Knightly E. QoS-driven server migration for internet data centers. *Tenth International Workshop on Quality of Service* (*IWQoS 2002*), Miami Beach, FL, U.S.A., 15–17 May 2002; 3–12.
43. Urgaonkar B, Shenoy P, Chandra A, Goyal P, Wood T. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Adaptive and Autonomous Systems* (*TAAS*) 2008; **3**(1):1–39.
44. Banga G, Druschel P, Mogul JC. Resource containers: A new facility for resource management in server systems. *Third Symposium on Operating Systems Design and Implementation* (*OSDI'99*), New Orleans, LA, U.S.A., 22–25 February 1999; 45–58.
45. Liu X, Zhu X, Singhal S, Arlitt M. Adaptive entitlement control to resource containers on shared servers. *Ninth IFIP/IEEE International Symposium on Integrated Network Management* (*IM 2005*), Nice, France, 15–19 May 2005.
46. Aron M, Druschel P, Zwaenepoel W. Cluster reserves: A mechanism for resource management in cluster-based network servers. *ACM SIGMETRICS 2000 International Conference on Measurement and Modeling of Computer Systems*, Santa Clara, CA, U.S.A., 17–21 June 2000; 90–101.
47. Chandra A, Gong W, Shenoy P. Dynamic resource allocation for shared data centers using online measurements. *Eleventh International Workshop on Quality of Service* (*IWQoS 2003*), Berkeley, CA, U.S.A., 2–4 June 2003; 381–400.
48. Chandra A, Pradhan P, Tewari R, Sahu S, Shenoy P. An observation-based approach towards self-managing web servers. *Computer Communications* 2006; **29**(8):1174–1188.
49. Chase J, Anderson D, Thakar P, Vahdat A, Doyle R. Managing energy and server resources in hosting centers. *Eighth ACM Symposium on Operating Systems Principles* (*SOSP'01*), Banff, Canada, 21–24 October 2001; 103–116.
50. Chase J, Grit L, Irwin D, Moore J, Sprenkle S. Dynamic virtual clusters in a grid site manager. *Twelfth International Symposium on High Performance Distributed Computing* (*HPDC-12*), Seattle, WA, U.S.A., 22–24 June 2003; 90–100.
51. Urgaonkar B, Shenoy P, Roscoe T. Resource overbooking and application profiling in a shared internet hosting platform. *ACM Transactions on Internet Technology* (*TOIT*) 2009; **9**(1):1–45. Article 1.
52. Urgaonkar B, Shenoy P. Sharc: Managing CPU and network bandwidth in shared clusters. *IEEE Transactions on Parallel and Distributed Systems* 2004; **15**(1):2–17.
53. Abrahao B, Almeida V, Almeida J, Zhang A, Beyer D, Safai F. Self-adaptive SLA-driven capacity management for internet services. *Tenth IEEE/IFIP Network Operations and Management Symposium* (*NOMS 2006*), Vancouver, Canada, 3–7 April 2006; 557–568.
54. Almeida J, Almeida V, Ardagna D, Francalanci C, Trubian M. Resource management in the autonomic service-oriented architecture. *Third International Conference on Autonomic Computing* (*ICAC'06*), Dublin, Ireland, 13–16 June 2006; 84–92.
55. Govindan S, Nath A, Das A, Urgaonkar B, Sivasubramaniam A. Xen and Co.: Communication-aware CPU scheduling for consolidated xen-based hosting platforms. *Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments* (*VEE'07*), San Diego, CA, U.S.A., 13–15 June 2007; 126–136.

56. Norris J, Coleman K, Fox A, Candea G. OnCall: Defeating spikes with a free-market application cluster. *First International Conference on Autonomic Computing* (*ICAC'04*), New York, U.S.A., 17–18 May 2004; 198–205.
57. Padala P, Shin K, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Salem K. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review* 2007; **41**(3):289–302.
58. Xu J, Zhao M, Fortes J, Carpenter R, Yousif M. On the use of fuzzy modeling in virtualized data center management. *Forth International Conference on Autonomic Computing*, *2007* (*ICAC 2007*), Jacksonville, FL, U.S.A., 11–15 June 2007; 25.
59. Karlsson M, Karamanolis C, Zhu X. An adaptive optimal controller for non-intrusive performance differentiation in computing services. *International Conference on Control and Automation* (*ICCA'05*), Budapest, Hungary, 26–29 June 2005.
60. Lu C, Abdelzaher T, Stankovic J, Son S. A feedback control approach for guaranteeing relative delays in web servers. *IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, May 30–June 1, 2001; 51–62.
61. Doyle R, Chase J, Asad O, Jin W, Vahdat A. Model-based resource provisioning in a web service utility. *Forth Symposium on Internet Technologies and Systems* (*USITS'03*), Seattle, WA, U.S.A., 26–28 March 2003.
62. Liu Z, Squillante M, Wolf J. On maximizing dervice-level-agreement profits. *Third ACM Conference on Electronic Commerce* (*EC 2001*), Tampa, FL, U.S.A., 14–17 October 2001; 213–223.
63. Abdelzaher T, Bhatti N. Web content adaptation to improve server overload behavior. *Computer Networks* 1999; **31**(11–16): 1563–1577.
64. Chandra S, Ellis C, Vahdat A. Differentiated multimedia web services using quality aware transcoding. *Nineteeth IEEE INFOCOM 2000*. Tel Aviv: Israel, 26–30 March 2000; 961–969.
65. Abdelzaher T, Shin K, Bhatti N. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems* 2002; **13**(1):80–96.
66. Li Z, Levy D, Chen S, Zic J. Explicitly controlling the fair service for busy web servers. *2007 Australian Software Engineering Conference* (*ASWEC'07*), Melbourne, Australia, 10–13 April 2007; 159–168.
67. Menasce D, Almeida V, Dowdy L. *Performance by Design*: *Computer Capacity Planning by Example*. Prentice-Hall: New Jersey, 2004.
68. Karve A, Kimbrel T, Pacifici G, Spreitzer M, Steinder M, Sviridenko M, Tantawi A. Dynamic placement for clustered web applications. *15th International World Wide Web Conference* (*WWW'06*). Edinburgh: Scotland, 23–26 May 2006; 595–604.