Information Sciences 199 (2012) 58–77

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Adaptive admission control algorithm in a QoS-aware Web system

Katja Gilly [a,*], Carlos Juiz [b], Nigel Thomas [c], Ramon Puigjaner [b]

[a] Departamento de Física y Arquitectura de Computadores, Miguel Hernández University, Elche, Spain
[b] Departament de Ciències Matemàtiques i Informàtica, University of Balearic Islands, Palma de Mallorca, Spain
[c] School of Computing Science, Newcastle University, Newcastle upon Tyne, UK

ARTICLE INFO

Article history:
Received 10 October 2009
Received in revised form 7 February 2012
Accepted 13 February 2012
Available online 28 February 2012

Keywords:
Admission control
Resource allocation
Load balancing
Overloaded Web server
Network monitoring
Performance evaluation

ABSTRACT

Internet traffic tends to show significant growth of demand at certain times of the day, or in response to special events. The consequence of these traffic peaks is that Web systems that are responding to user demands are congested due to their inability to serve a large volume of requests. The case for admission control in these situations is even stronger when Quality of Service (QoS) is considered as a primary objective in the Web system. In this work, we address two issues: on one hand, we consider and compare five throughput predictors to be used in a Web system in order to track its performance and, on the other hand, we propose a QoS-aware admission control and load balancing algorithm that prevents the Web system from sudden overload. The admission control algorithm is based on a resource allocation scheme that includes a throughput predictor. In order to obtain a low overhead, the monitoring of traffic arriving at the Web system is performed following an adaptive time slot scheduling based on the burstiness factor that we defined in previous work. Results show the benefits of our adaptive time slot scheduling compared to a fixed time scheduling. A discussion of the results of the five throughput predictors and the admission control algorithm is provided. We also compare the performance of our algorithm with Intelligent Queue-based Request Dispatcher (IQRD). The algorithm is designed to be included in a Web system composed by a set of Web servers distributed locally, which can also form part of a wider geographically distributed load balancing architecture.

1. Introduction

For Web services to satisfy the demands of end users, they must continue to offer a good level of performance, even in the fact of erratic and unpredictable demands. It has been widely observed that Internet traffic is self similar and that sudden bursts of packets can reach a point in a network infrastructure that offers Web services. This affects the performance of the system if it is not able to process that increase in the demand. High variance in incoming traffic and service time distributions can collapse the system in few seconds; therefore it is necessary to control these features by an adaptive algorithm. We propose an adaptive admission control algorithm that prevents the system from a sudden overload by predicting the throughput of the Web servers. Five different throughput predictors are considered in this work.

The problem of allocating resources to a Web System that considers QoS is also addressed in this work. We have considered a Web system that is composed of a cluster of Web servers, as shown in Fig. 1. The algorithm includes adaptive resource

Abbreviations: App/DB, application/database; AS, autonomous system; CPU, central processing unit; HTTP, hypertext transfer protocol; IQRD, intelligent queue-based request dispatcher; LMS, least mean square; NLMS, normalised least mean square; RR, round robin; SLA, service level agreement; TCP, transmission control protocol; OS, operative system; QoS, quality of service; URI, uniform resource identifier.

* Corresponding author. Tel.: +34 96665 8565; fax: +34 96665 8814.
E-mail addresses: katya@umh.es (K. Gilly), cjuiz@uib.es (C. Juiz), Nigel.Thomas@ncl.ac.uk (N. Thomas), putxi@uib.es (R. Puigjaner).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2012.02.018

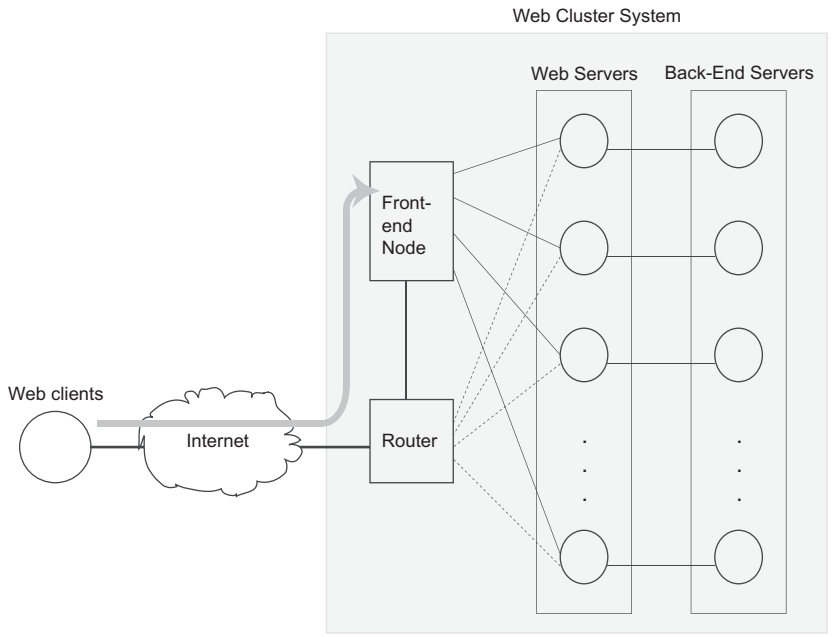

Fig. 1. The Web architecture is made up of several mirrored Web servers and their corresponding database servers. The model architecture is one-way, which means that the incoming HTTP requests go through the front-end node but their HTTP responses use a different way to prevent a system bottleneck in this node.

allocation for each server, considering the values of certain monitored performance metrics that allow the algorithm to learn the current state of the Web system. Predictions of throughput and utilisation are computed based on these metrics and, in case future congestion in the Web system is forecast, the admission control part of the algorithm is initiated. Hence, overload situations can be addressed by this algorithm to guarantee satisfactory performance of the system by controlling the utilisation level of the servers within the cluster.

An important contribution of our work is the adaptive overhead our solution introduces in the system. It is critical to avoid checking the system continuously, e.g. at each incoming request, because this produces an enormous overhead in the front end of the system. It is also risky to check the system during fixed intervals because a sudden increase may not be detected until the system is already overloaded. We have analysed the most important related work in order to learn how other authors handle or control overhead, and we have observed that very few works propose methods to reduce the overhead.

The following sections of this paper are organised as follows:

- Section 2 describes related studies on admission control and includes a table that sums up the characteristics of the proposals we are principally interested in.
- The steps we take in order to obtain a low overhead are detailed in Section 3.
- Section 4 introduces an overview of the algorithm that includes the system architecture details, the QoS and the metrics considered by the algorithm.
- The throughput predictors we propose to be included in the algorithm are described in Section 5.
- The resource allocation strategy and load balancing policy are in Section 6.
- In Section 7, we have included a description about the workload used in the simulations and the load balancing policy applied.
- Simulation results of the comparison of the different throughput predictors are included in Section 8.
- In order to compare our adaptive time slot scheduling to a fixed time slot scheduling, we have run a set of simulations whose results are included in Section 9.
- We present some results from an implementation of Intelligent Queue-based Request Dispatcher, the algorithm proposed by Sharifian et al. in [41], and comment on the performance differences detected in comparison to our algorithm in Section 10.
- Finally, we discuss some concluding points and the open problems.

2. Related work

The problem of the admission control in a Web server has been widely addressed in literature. In this Section, we introduce the most significant Web admission control related algorithms and organise them in Table 1, where we include a summary of the characteristics we consider the most important from the cited admission control proposals.

Table 1
Main characteristics of admission control policies: (0) Reference; (1) The year the reference is published. (2) QoS-aware algorithm (Y = yes, N = no). (3) Classic control theory-based. (4) Session-aware. (5) Cluster of Web servers considered. (6) Performance metric monitored (1 = CPU utilisation, 2 = service or response time, 3 = number of requests, 4 = arrival rate, 5 = transaction size, 6 = number of processes running in the Web server). (7) Invocation frequency (P = Periodical, NP = Non Periodical).

(0) Ref.	(1) Year	(2) QoS-aware	(3) Feedback control theory	(4) Session aware	(5) Web Cluster	(6) Performance metric	(7) Invocation frequency
[4]	98	Y	N	N	N	2, 6	NP
[37]	98	Y	N	N	Y	1–3, 5	NP
[11]	99	Y	N	N	Y	3	NP
[17]	99	Y	N	N	N	6	NP
[27]	00	Y	N	Y	Y	2, 4	NP
[3]	02	Y	Y	N	N	1	P
[9]	02	Y	N	Y	Y	1, 4	P
[16]	02	Y	N	Y	N	1	P
[30]	02	Y	N	Y	Y	2	NP
[44]	02	Y	Y	N	N	1, 4	NP
[13]	03	Y	N	N	N	2	NP
[12]	03	Y	N	Y	N	2, 4	NP
[14]	03	Y	N	N	N	3	NP
[45]	03	Y	N	N	N	2	P
[18]	04	Y	N	Y	N	1–2	NP
[5]	05	Y	N	N	N	1–2, 4	NP
[40]	06	N	N	N	N	5	NP
[48]	06	N	N	N	Y	1–3, 5	NP
[29]	08	N	Y	N	N	1	P
[41]	08	Y	N	N	Y	1–3	P
[43]	08	Y	N	N	Y	2–4	NP
[10]	09	Y	N	Y	Y	2, 4	NP
[39]	09	Y	N	Y	Y	3	P

Starting in chronological order, there are several pioneering *QoS-aware scheduling policies* that should be mentioned in this section, despite including no explicit admission control mechanism. In particular, we refer to the work of Almeida et al. [4], Pandey et al. [37] and Eggert and Heidemann [17]. These authors have developed priority-based scheduling methods that permit differentiation between several types of traffic in the Web server with modifications to the Web server [37,17], or both the Web server and the OS kernel [4]. Most of the works considered in this section are QoS-aware, as indicated in column (2) of Table 1. Hence, let us consider other characteristics in order to group the proposals.

Feedback control theory has been used by some authors in admission control algorithms. Abdelzaher et al. [3] propose a mechanism based on a utilisation control loop that also permits guaranteeing individual time constraints. By contrast, Voigt and Gunningberg [44] propose the maximum rate of accepted requests as the performance metric to be controlled. A non-QoS-aware admission control mechanism based on discrete-time control theory was implemented in the Apache Web server by Kihl et al. [29]. In this case, the monitored metric is the server Central Processing Unit (CPU) utilisation. The use of control theory in the proposals is reflected in column (3) of Table 1. This table also includes information about the monitored metrics (in column (6)), that include the CPU utilisation in these last three cases.

Other works describe proposals that are defined for *e-commerce environments*, and hence, consider session-based workload. Kanodia and Knightly [27] develop a multi-class session admission control based on envelopes (that describe class and service demands) in order to guarantee class-based target delays. Also controlling class processing delays, Kihl and Widell [30] propose a session-aware admission control algorithm for distributed Web sites. There are some other proposals that also include *a set of Web servers* (see column (5) of Table 1) as the load balancing and admission control scheme for a Web-cluster. In Aweya et al. [9], the acceptance rate of client requests is adaptively set based on the CPU performance measures and is not dependent on the request class.

The selection of the sessions to discard in a congestion situation depends on the approach used. As longer sessions in an e-commerce environment are normally the ones that result in a purchase, in order to guarantee the completion of these sessions, Cherkasova and Phaal [16] propose two admission control mechanisms also based on the CPU utilisation. However, Chen and Mohapatra [12] discriminate against the sessions that are likely to get aborted due to overload, prioritising the ones that have a higher probability of completion.

Several self-configuring session-based admission control algorithms have been proposed recently. Bartolini et al. [10] describe a self-adjusting policy that switches between two modes depending on the arrival rate detected at the system. When the system is not overloaded, their approach takes admission control decisions at fixed time intervals. If the arrival rate exceeds a limit, then the admission control decisions are taken each time a new session arrives to the system. A self-adaptive architecture that learns user behaviour and predictes the purchase probability of Web sessions is described by Poggi et al. [39]. A method that does not require any modification in either in the Operative System (OS) or the Web software (Web server, application server or database) is presented in [18] by Elnikety et al. Instead, they describe a transparent proxy that intercepts the requests and measures their service times.

In column (4) of Table 1, we have included the use of *session-based* or *request-based workload* in the cited admission control proposals. When considering request-based admission control policies, some authors introduce thresholds that activate the admission control to guarantee the performance. These thresholds can depend on the number of requests queued in a server [11], the execution time of requests [48,5], or a combination of several metrics [43,14]. Urgaonkar and Shenoy [43] propose a very interesting low-overhead admission control algorithm for handling extreme overloads. In this work, the authors introduce some dynamically adjusted class-based delays to invoke the admission control algorithm and vary these delays with the workload changes when the system is not overloaded. They switch the admission control policy to a periodical threshold-based strategy when the load increases.

When defining the trigger for the admission control policy, its value can be static (as it is in the case of [11]) or dynamic to the status changes of the Web system [48,14,43]. Considering proposals that adapt the admission control policy to system changes, Chen et al. [13], describe an admission control algorithm that includes differentiated services. They try to guarantee a maximum response time by adjusting periodically the resource allocation of the system on the basis of a service time prediction for each request class. Sharifian et al. in [41], describe an adaptive load balancing algorithm that includes an admission control policy for a cluster of Web servers. Requests are differentiated depending on their service times and the load of the Web server is estimated using a queueing model based on performance metrics that are obtained periodically. Rejection starts when there is no remaining capacity for a specific class of requests.

Although Web services are beyond the scope of this paper, there are other QoS-aware works that should be mentioned. For instance He et al. [25], who define an agent-based framework for Service Level Agreement (SLA) management. Reliability is also considered very important in this environment. Oh et al. [35] describe a method to execute collaborative business processes within time constraints is developed. Another example is an approach that splits the Web services in several stages, each of them with a specific admission controller, included in [45].

We have tried to organise previous admission control proposals in Table 1, that sums up some of the characteristics of these previous works. Each row of the table corresponds to a reference (in chronological order). Normally, the invocation frequency of an algorithm (detailed in column (7)) is the same as the monitoring frequency of the performance metric that is used as the input of the algorithm. This performance metric is needed to take admission control decisions and can be demanded in different ways depending on the proposal. In Table 1, we can observe that some authors define a fixed time interval to obtain monitored performance values and invoke the admission control algorithm [3,9,16,45,41,29,39], while others check the performance metric selected and then execute the admission control algorithm when a determined event has occurred, i.e. each time a new request or session arrives to the Web system [4,37,11,17,27,30,44,13,12,14,18,5,40,48]. Adaptive invocation is only included in two recent works [43,10], as they introduce a dynamic variation of the invocation times of the admission control algorithm that depend on the workload with the goal of overhead reduction. In both cases, there is a switch in the admission control policy depending on the overload of the Web system. We have designed an algorithm that also schedules its invocation times adaptively in order to reduce the overhead, as described in the next section.

3. Aiming for low overhead

We propose an admission control and load balancing algorithm that is based on throughput prediction for a Web system. In order to achieve a low overhead of the algorithm, we plan the invocation times based on the arrival rate. In [21] we defined a burstiness factor and, based on this factor, we proposed an adaptive time slot scheduling that sets the frequency that the algorithm is invoked. Each execution of the algorithm needs some monitoring metrics that are collected from the server nodes. In order to reduce the network traffic in the cluster, the monitoring metrics are sent to the distributor node at the algorithm invocation times.

Let us describe briefly how the adaptive time slot scheduling works (more information can be found in [21]). We divide the time into slots (k) of different durations ($d(k)$) during the experiment. The burstiness factor, $b(k)$, varies its value in a range of $[0,1]$ and gives an indication of the burstiness perceived in the entry point of the system. The burstiness detected in the system influences the execution of the algorithm in this way; when an increase in the burstiness is detected, then the algorithm is invoked more frequently, and viceversa. Therefore, the algorithm is executed more often when there is a high demand of Web services, trying to avoid a sudden overload in the Web system, and viceversa when demand is low.

In general, there are two options to invoke an algorithm considering the works discussed in Section 2. Supposing the cost of executing the algorithm is $O(x)$ and the total observation time is T, let us then analyse the cost of each option:

- *Periodical invocation frequency*: In this case, the algorithm is executed periodically and the decisions taken are maintained until the next execution of the algorithm. The total observation time T can then be divide into n periods or slots. The approximate cost of this case would be $O(n \cdot x)$.
- *Non-periodical invocation frequency*: Normally, this means that the algorithm is executed each time a new request or session arrives to the system. In this case, and considering that r requests are received during the whole observation time, the cost of the algorithm would be $O(r \cdot x)$, r being several orders of magnitude greater than n. **Adaptive scheduling** can also be included as non-periodical scheduling. This is the case of [43,10] that vary the invocation method depending on the load, but only under some circumstances, as we have previously commented on in Section 2.

We propose a variation on the frequency of the algorithm's execution in all cases, always depending on the burstiness detected in the system. Let us introduce now the algorithm overview in order to describe the scenario and metrics considered in our proposal, and how we define the QoS in this scope.

4. Algorithm overview

Our algorithm is adaptive because it is invoked depending on the arrival rate as we have described in the previous section. Each time it is invoked, some computations need to be performed in order to take the admission control and load balancing decisions that will remain till the next invocation. The period of time between invocations is considered a slot.

Fig. 2 depicts the general steps that are taken by the algorithm. Once the burstiness factor has been computed and the monitored throughput has been retrieved, CPU utilisation and service time values from the server nodes, we predict the throughput the nodes will get during the next slot. We define five throughput predictors in Section 5 that give us the trend of the system behaviour. These predictors permit the algorithm to take decisions about the distribution of the load in the Web system to maintain the performance of the system, as it will be described in the next section. We introduce the resource allocation policy in Section 6, where we describe how the utilisation of the server nodes is assigned to attend each class of requests that may arrive to the Web system.

An overview of some of the characteristics of the algorithm is included in this section. First, we introduce the system architecture that is proposed for the algorithm. We also describe the QoS considered in the system by defining the Service Level Agreement of the requests. And finally, we discuss the performance metrics obtained by the algorithm.

4.1. System architecture

The system architecture proposed is based on Web cluster-based network servers and includes a front-end Web switch. A layer-7 Web switch is normally described as a content-aware switch that can de-encapsulate the requests up to the application level and classify them on the basis of this information, but it can easily be the bottleneck of the Web system. Other authors [8,38,28,15,47] have already solved this problem by transferring the request distribution mechanism to the back-end nodes and replacing the content-aware Web switch with a content-blind Web switch.

Another problem of a content-aware Web switch is the distribution of requests based on (HTTP)/1.1 persistent connections, that has also been solved in previous proposals by other authors [7,31,42,32,33]. Hence, we consider that our one-way content-aware architecture includes one or more distributor nodes, leaving the underlying Transmission Control Protocol (TCP) implementation open to implement one of these proposals.

The cluster of Web servers is locally connected to the Web switch in a two-tier organisation (Web server and Application/Database (App/DB) server), as shown in Fig. 1. We have considered 5 sets of Web and App/DB servers. Each Web server attends the requests that ask for static files, namely static requests and the App/DB server is accessed when the request asks for a Web page that needs to retrieve dynamic content (dynamic requests).

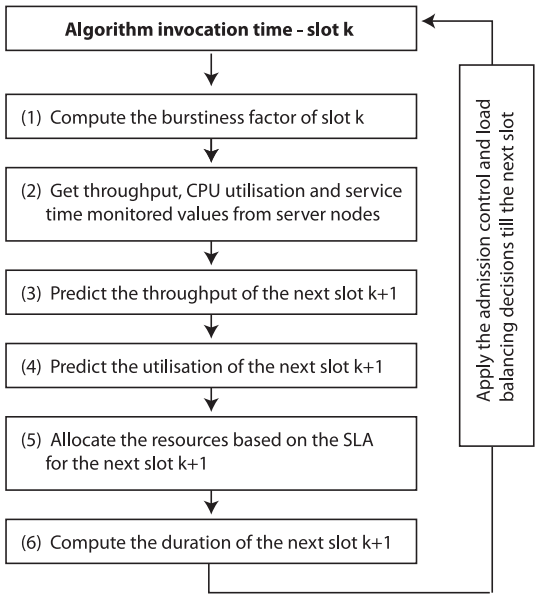

Fig. 2. Adaptive admission control and load balancing algorithm overview.

The Web cluster can be considered as either a part of a wider system that geographically distributes the load or as an Autonomous System (AS).

4.2. QoS-awareness

Different classes of requests are distinguished in the algorithm. We have already described the differentiation between static and dynamic requests as they result in different loads in the two tiers of the server architecture. A QoS attribute in the requests is also introduced. This attribute represents the priority of each request and can be considered either implicit in the content of the HyperText Transfer Protocol request (i.e. in the Uniform Resource Identifier (URL)) or determined by the client profile in an e-commerce environment. Depending on the priority of each request, we set a fraction of the utilisation of the whole Web system to be used by that request class. Therefore, we consider a set of classes, $C = \{c_1, c_2, \ldots, c_r\}$, and define for them a normalised utilisation value in a decreasing order. Hence, the class of requests that represent c_1 have a higher priority than the class c_2, and so on. The sum of the utilisation values of all the classes is equal to 1, which represents the whole utilisation of the Web system. The admission control and load balancing algorithm adaptively defines how the different classes of requests are distributed among the server nodes.

4.3. Performance metrics used in the algorithm

It is important to detect the resource of the Web system that can potentially be the bottleneck and hence, determine the performance metrics that manage the admission control and load balancing algorithm. As we have previously described in Section 2, many authors have considered the CPU utilisation as a metric to be checked in the admission control mechanism [37,3,9,16,44,18,5,48,29,41]. It has been detected that the main bottleneck of a Web system is the database server when dynamic content is requested [40]. As it is out of the scope of this paper to study the scheduling in databases and the times the transactions spend waiting for locks [34], we have considered the CPU utilisation as the main metric to estimate in the Web and App/DB server nodes.

Some performance metrics are monitored in order to estimate the Central Processing Unit utilisation; some of them can be obtained from the front-end of the Web system, but others have to be requested from the servers. The monitoring results are transferred to the distributor node(s) when the admission control and load balancing algorithm is invoked.

Let us describe the metrics that we need to monitor to execute our algorithm:

- The arrival rate, that is used to compute the burstiness factor, and hence to set the adaptive time slot scheduling, can be easily monitored by the front-end of the Web system by counting the requests that arrive during a slot. The arrival rate for a slot k, $\lambda(k)$, is obtained by dividing the total number of incoming request by the slot duration, $d(k)$.
- The average throughput for a slot k, $x(k)$, is obtained from the Web and Application/Database servers, and used to estimate the throughput of the Web and Application/Database servers during next slot. The throughput is also used to control the error in the throughput prediction.
- The service times needed to process the static and dynamic requests are also obtained from the Web and Application/Database servers. The average service time at each slot, $\delta(k)$, is used to estimate the utilisation of the Web and Application/Database servers as it is known that the service times increase when the server starts to be overloaded [13,18].
- The average CPU utilisation of the Web and Application/Database servers, $u(k)$, is used as a factor in the expression of some throughput predictors, and also used to control the error of our CPU utilisation prediction.

5. Throughput prediction

We compute the throughput prediction in order to obtain an estimation of the behaviour of the Web system after obtaining the burstiness factor and the collection of the monitored metrics form the server nodes, as it is depicted in Fig. 2. Thus, in this section, the five throughput predictors considered in this study are introduced. A previous version of the predictors P1-P3 was introduced in [20]. We have extended the research in throughput prediction in order to achieve better results. Some of these predictors were also presented in a short paper format in [22]. Let us introduce these five predictors.

5.1. P1: based on filtering

This predictor is a moving average between the estimated value of the throughput in the last slot and the harmonic mean of the real throughput measured in the last two slots.

$$\hat{x}_1(k+1) = (1 - a(k+1)) \cdot \hat{x}_1(k) + a(k+1) \cdot \frac{2}{\frac{1}{x(k)} + \frac{1}{x(k-1)}} \tag{1}$$

$a(k+1)$ being the probability that balances the weight of the expression in function of the duration of the next slot, $d(k+1)$; hence, it indirectly depends on the burstiness factor:

$$a(k+1) = \frac{2 \cdot T - d(k+1)}{2 \cdot T + d(k+1)}$$

Therefore, this estimated throughput depends on two terms; the last computed throughput prediction and the average of the actual and previous throughput measurements. The result of this computation is filtering throughput values based on the probability $a(k+1)$. Since the duration of the slot $k+1$ depends on burstiness, the weight of $a(k+1)$ indirectly depends on burstiness too. The factor $a(k+1)$ normally places more weight upon previous throughput measured in the servers during the slots k and $k-1$ than on the previous throughput estimation, unless the value of $d(k+1) > \frac{2}{3} \cdot T$. The smaller the duration of the slot, the more weight the previous throughput measures have in this predictor. The main effect of this estimation is smoothing traffic peaks in order to hold an accurate performance estimation of the servers in the long run.

5.2. P2: based on burstiness

This predictor includes the burstiness factor we described in [21]. It is included in order to factorise the tendency of the burstiness during the last two periods with the throughput variation as a factor $\beta(k+1)$:

$$\beta(k+1) = (b(k) - b(k-1)) \cdot |x(k) - x(k-1)| \tag{2}$$

It measures the difference between the current and the previous burstiness factors, and then multiplies it by the absolute value of the difference between measured throughputs at slots k and $k-1$. The resulting product expresses the amount of variation of the servers' performance due to the burstiness on client transaction arrivals.

This factor is then multiplied by the server utilisation in order to scale the increase or decrease of previous slot throughput:

$$\hat{x}_2(k+1) = \hat{x}_2(k) - (\beta(k+1) \cdot u(k)) \tag{3}$$

The goal of this estimator is to decrease the throughput prediction when a burst of requests is detected, and viceversa, depending on the current utilisation of the servers.

5.3. P3: based on filtering and burstiness

The harmonic mean of the predictor P1 and P2 is considered as the third predictor in order to balance the effect predictors P1 and P2 may have in the system.

$$\hat{x}_3(k+1) = \frac{2}{\frac{1}{\hat{x}_1(k+1)} + \frac{1}{\hat{x}_2(k+1)}} \tag{4}$$

5.4. P4: based on Least Mean Square (LMS)

We have also considered the LMS algorithm [46,24] to predict the throughput. LMS introduces an iterative procedure that makes successive corrections to a weight vector that minimises the mean square error. This filter has been previously used in throughput prediction by Garroppo et al. in [19].

Let $\underline{w}(k+1)$ denote the weight vector of the LMS filter that is computed at the kth slot. The operation can be expressed by the following recursive operation:

$$\underline{w}(k+1) = \underline{w}(k) + \mu \cdot [x(k) - \hat{x}_4(k)] \cdot \underline{x}(k) \tag{5}$$

where μ is the step-size parameter. The vectors $\underline{w}(k)$ and $\underline{x}(k)$ are defined as the following expressions, being M the number of tap weights used in the adaptive transversal filter:

$$\underline{w}(k) = [w_0(k), w_1(k), \ldots, w_{M-1}(k)]^T$$
$$\underline{x}(k) = [x(k), x(k-1), \ldots, x(k-M+1)]^T$$

The predicted value of throughput is obtained by linear prediction with this expression:

$$\hat{x}_4(k+1) = \sum_{x=0}^{M-1} w(x) \cdot x(k-x) \tag{6}$$

5.5. P5: based on Normalised Least Mean Square (NLMS)

A normalised version of the LMS filter was proposed in [23] to avoid the sensitivity of the LMS algorithm to the scaling of its input $x(k)$. The solution is to normalise the previous expression by dividing the vector $\underline{x}(k)$ by the square of its Euclidean norm.

$$\underline{w}(k+1) = \underline{w}(k) + \mu \cdot [x(k) - \hat{x}_5(k)] \cdot \frac{\underline{x}(k)}{\|\underline{x}(k)\|^2} \tag{7}$$

6. Resource allocation and load balancing

Throughput prediction enables us to estimate the maximum utilisation allowed in the servers for each service class, and then to set a maximum number of accepted requests in order to limit the utilisation of the servers and avoid a possible congestion situation, while guaranteeing the QoS.

With the value of one of the throughput predictors described in previous section, $\hat{x}(k+1)$, and the service time, $\delta(k)$, obtained from the servers we can make an approximation of the utilisation level that the servers will have in the following slot for each class of service. An adjustment of these utilisation levels is necessary in order to fulfil the SLA. Once we obtain the adjusted utilisation level the servers should not exceed, we can calculate the maximum number of requests each server can process of each class of service.

Hence, as a first step, we predict the utilisation that each class of traffic will have in each server of the Web system with this expression:

$$\hat{u}(k+1) = \hat{x}(k+1) \cdot \delta(k) \tag{8}$$

This prediction does not include the SLA we have defined for each class of requests. We then normalise the predicted utilisation to guarantee the priority requirements of each class. We need to add some subscripts to the expressions because the throughput and utilisation predictions are calculated for each Web and App/DB server, and for each class of service. Therefore, we include three subscripts:

- The first subscript indicates the number of the Web server and App/DB server set, that is represented by $i \in \{1,\ldots,N\}$.
- The second subscript states the class of service and it is represented by $j \in \{1,\ldots,r\}$.
- The third subscript points out whether the content requested is static or dynamic, $z \in \{sta, dyn\}$. In case the content is static, then it will be attended by the Web server. The App/DB server will carry out most of the processing work if the content requested is dynamic. Hence, we compute the utilisation separately in the two tiers of the Web system architecture.

The normalisation of the utilisation estimation is done to distribute 100% of the available capacity of the Web system between all the classes of requests, where N is the number of Web and App/DB server sets that are included in the Web system:

$$\hat{u}'_{i,j,z}(k+1) = (c_j \cdot N)\frac{\hat{u}_{i,j,z}(k)}{\sum_{\forall i}\hat{u}_{i,j,z}(k)} \tag{9}$$

With this normalisation, we ensure that each traffic class has reserved the utilisation of the Web system that corresponds to its SLA. In order to know the maximum number of requests that are going to be accepted for each traffic class at each server during the following slot, we make the inverse operation and multiply the obtained throughput by the duration of the next slot:

$$\gamma_{i,j,z}(k+1) = \frac{\hat{u}'_{i,j,z}(k+1)}{\delta_{i,z}(k)} \cdot d(k+1) \tag{10}$$

During the next slot, the distributor counts the number of accepted requests of each class in each Web and App/DB server. When $\gamma_{i,j,z}(k+1)$ is reached, the server stops attending to that class of requests until the next invocation of the admission control and load balancing algorithm.

Regarding the load balancing, we have designed a dispatching mechanism that is based on the Round Robin (RR) classic load balancing policy to distribute the incoming requests among the Web and App/DB servers, distinguishing among the different classes of service requested. The distributor node performs a content aware analysis to learn which service class is needed to attend each request and organises the Web and App/DB servers in a cyclical manner for each class of service. So, if there are N servers and all the servers are *active* for attending requests that ask for a particular class of service, each server $i \in 1,\ldots,N$ will receive the request number x, if $i = (x \bmod N) + 1$.

Note that it is possible that not all the servers are *active*. If a Web server or App/DB server reaches the maximum number of requests for a class of service (j) that was set in the previous invocation of the admission control algorithm, then it is *disabled* from the possible selection by the distributor node until the next invocation of the algorithm. Therefore, the rejection of requests starts when all the Web or App/DB servers of the system are disabled for a class of service.

7. Simulation scenario and workload

We have implemented our algorithm in the simulation tool OPNET Modeler [1] which facilitates accurate simulation of the layers of the TCP/IP stack. Hence, a realistic simulation model has been considered, including all the protocols below, in the TCP/IP stack that take place during an application-application level communication. The routing protocol considered at

Table 2
Workload 1 specification (for each service) for the throughput prediction comparison.

Number of Web servers	5
Number of clients	30, 40, 50, 60, 70, 80, 90, 100
File size	Lognormal body and heavy-tailed
User think time (s)	Pareto ($k = 0.3, \alpha = 1.4$)
User session duration (s)	Exponential ($\mu = 600$)
Session inter-repetition time (s)	Exponential ($\mu = 30$)

network level is RIP in the routers of the network. All server nodes of the simulated scenario are modelled as Sun Ultra 10,333 MHz workstations running Solaris OS. Their cache memory has been disabled. We have made modifications to the TCP OPNET standard model to adapt it to the TCP hand-off mechanism [26,36].

We consider two different service classes, named c_1 and c_2, in all the simulations. Each service class contains two types of applications: one that asks for dynamic content and another that asks for static content. Static requests are attended by the Web servers while dynamic requests require access to the App/DB server. Hence, we define in the system four types of applications or services (c_1 static, c_1 dynamic, c_2 static and c_2 dynamic) that execute concurrently in the Web system and whose service is asked, also concurrently, by the Web clients. The SLA specified for each service class is: $c_1 = 0.625$ for class-1 requests and $c_2 = 0.375$ for class-2. Requests asking for both classes of service are introduced in the system in the same proportion. We refer indistinctly to traffic class, service class or request type in order to indicate a class of service. Each simulation has been repeated four times with four different seeds during a total simulation time of $T = 2000$ s.

In order to test the admission control algorithm, we need to overload the system. Five Web and App/DB servers are included in the Web system to attend the two different classes of requests, c_1 and c_2. The workload is generated in the Web system by 30, 40, 50, 60, 70, 80, 90 and 100 Web clients, as we are interested in stressing the system to test the algorithm with an increasingly high workload. Hence, the Web system starts rejecting requests when it is overloaded.

We configure two workloads in order to test the algorithm more accurately. Both are basically the same, the only difference is in the user think time. Let us describe these two workloads independently.

Workload 1: We consider a Pareto distributed user think time in the Web clients, and the session duration and the session inter-repetition time are modelled according to a exponential distribution. The file size has been obtained from the logs of the 24th of June of the 1998 World Cup Web Site [2]. Arlitt and Jin [6] estimate, after analysing these logs, that the body of the unique file size distribution follows a lognormal distribution and also that it is heavy-tailed.

Table 2 shows this workload specification, that is applied to each service demand. Hence, the Web clients ask for the four services defined concurrently. This implies more than 200 Web requests per seconds when 100 clients are active in the scenario, as it can be observed in Fig. 3, that are attended by the five server nodes defined in the cluster.

Workload 2: The second workload is basically the same as the first one, but with one difference. In order to configure a more variable workload, we insert traffic peaks every 300 s. Hence, we modify the user think time for 30 s in order to provide more burstiness in the simulation. After each 300 s of the workload specified in Table 2, we change for 30 s the user think time to Pareto ($k = 0.2, \alpha = 1.4$). This means a rise in the arrival rate after certain time of "stability" that permits us to test the algorithm under more severe circumstances. In Fig. 3 we can observe that the arrival rate increases up to 350 Web requests per second for 100 clients during these 30 s periods.

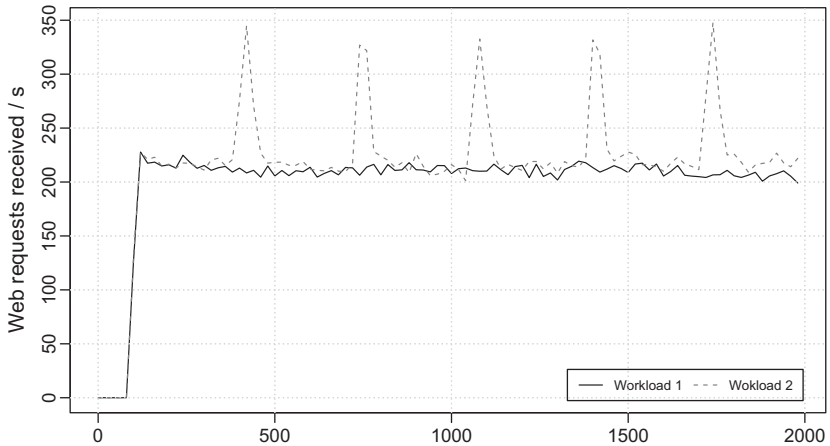

Fig. 3. Workload 1 and Workload 2 generated by 100 clients.

8. Throughput prediction comparison

The five throughput predictors described in Section 6 have been implemented and tested in our admission control and load balancing algorithm. As we introduce the same proportion of static and dynamic requests in the system and the admission control algorithm reserves some CPU utilisation for each service type, the first components of the Web system that become overloaded are the App/DB servers. Therefore, the bottleneck of the architecture are the App/DB servers, as we will observe in the results reported in the sections below.

In order to check this, we have to analyse the utilisation level results, but first, let us have a look at the error in throughput predictions.

8.1. Error in throughput predictions

We have computed the mean squared error of the throughput predicted in a slot and the throughput monitored in the next slot in order to compare the effectiveness of the predictions made by the five predictors.

$$E_{ms} = \sqrt{\frac{\sum_{k=1}^{n}(x_{i,j,z}(k) - \hat{x}_{i,j,z}(k))^2}{n}} \tag{11}$$

The computed error for each prediction is shown in Fig. 4 for static requests, and in Fig. 5 for dynamic traffic, and both workloads. We have used the same scale for both static and both dynamic traffic results in order to compare them.

Considering static requests, it can be observed that the error in the prediction grows with the number of clients in both cases (it increments more in the case of *Workload 2*), while for dynamic requests, the error is limited by the admission control algorithm. Hence, the prediction error is reduced at some points. It can be observed, in the upper part of Fig. 5a, that the throughput errors of all predictors for class-1 dynamic requests increase slightly until 70 clients and then, they are abruptly reduced to a value near to zero. This decreasing tendency is directly related to an increasing number of rejections carried out by the algorithm as the number of clients grows larger.

While requests are not rejected, the throughput prediction error tends to increase as the throughput increases. However, when rejections start, the value of the prediction error remains near zero because the throughput is controlled by the algorithm, hence, the predictions are more precise.

8.2. CPU utilisation

In order to analyse the CPU utilisation, we represent in Figs. 6 and 7 the 95th percentile of the average CPU utilisation of the Web servers and App/DB servers, respectively. It can be observed that the Web Servers are not overloaded whereas the App/DB servers reach their maximum utilisation with 50 (Fig. 7a) and 70 clients (Fig. 7b). We have omitted the results of *Workload 2* in Web servers because comparatively they increase slightly (up to 12%), but basically are very similar.

We can also observe that the average CPU utilisation achieved by the App/DB servers guarantees the SLA defined in the algorithm. For class-1 traffic, $c_1 = 0.625$ (62.5% of utilisation of the servers) and for class-2 $c_2 = 0.375$ (37.5%).

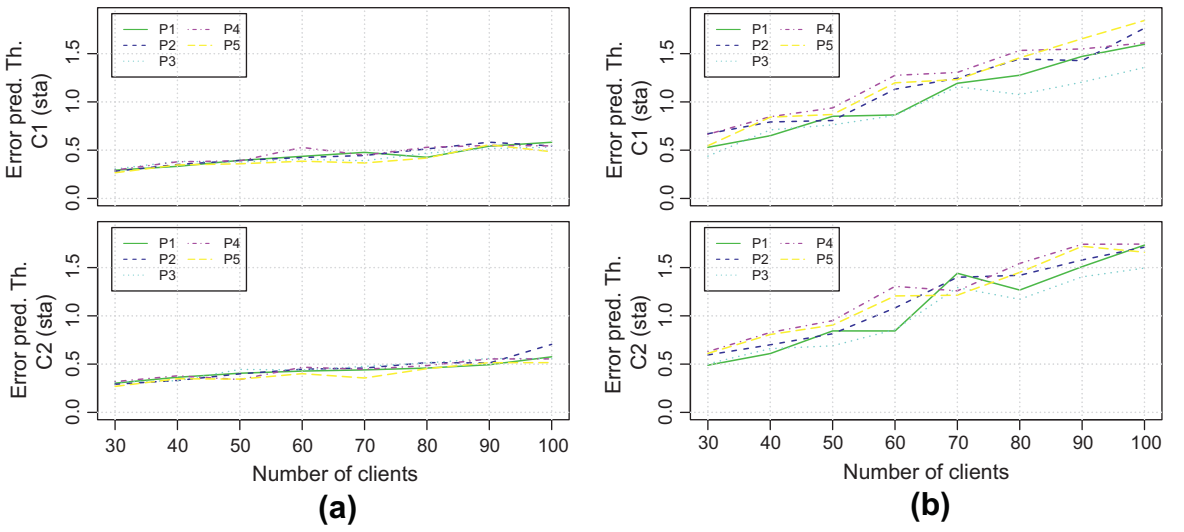

Fig. 4. Mean squared error of the throughput predictions of static requests: (a) Workload 1; (b) Workload 2.

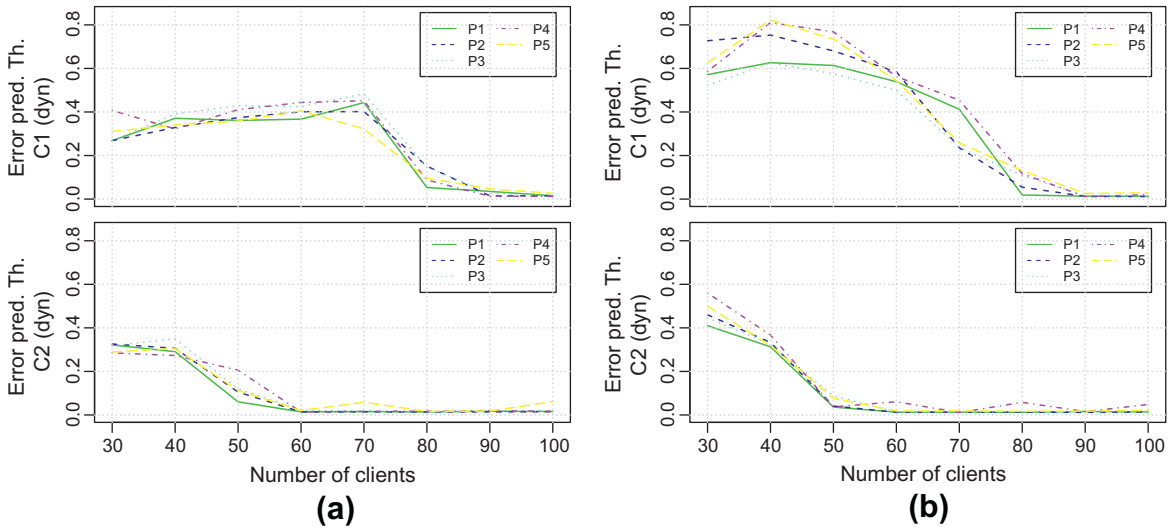

Fig. 5. Mean squared error of the throughput predictions of dynamic requests: (a) Workload 1; (b) Workload 2.

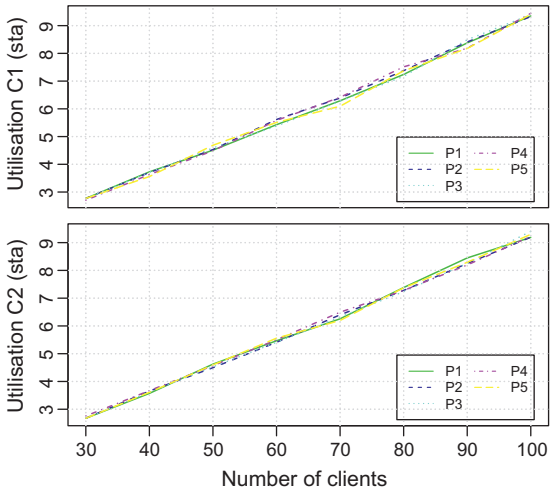

Fig. 6. Workload 1: 95th percentile of the average Web server utilisation for static requests.

8.3. Rejection of requests

An important consequence of the admission control algorithm is the rejection of requests when the Web system is overloaded. By observing Figs. 6 and 7 we can guess that rejections of class-1 dynamic requests start with 70 and 50 clients for *Workload 1* and *Workload 2*, respectively. Fig. 8, that represents the number of rejected requests, confirms this guess. We have not included the static requests as none of them is rejected in any of the simulations.

8.4. Response time

Fig. 9 represents the response time of static requests for both workloads. Despite no static request being rejected, the increase in the arrival rate that represents *Workload 2* has an impact on the response time of the static requests (Fig. 9b), compared to *Workload 1* (Fig. 9a).

The response time of dynamic requests, represented in Fig. 10, is more meaningful as the App/DB servers are congested with the increase of traffic. If we analyse the case of *Workload 1* in Fig. 10a, we can note some differences among the response time obtained by the predictors. Focusing on the last case, 100 clients, we can detect that the predictors P1, P2 and P3 obtain a higher response time than predictors P4 and P5. This is also depicted in Fig. 10b, which represents the response time for *Workload 2*. We can also observe that the maximum response time for both workloads is around 2.5 s, meaning that that our

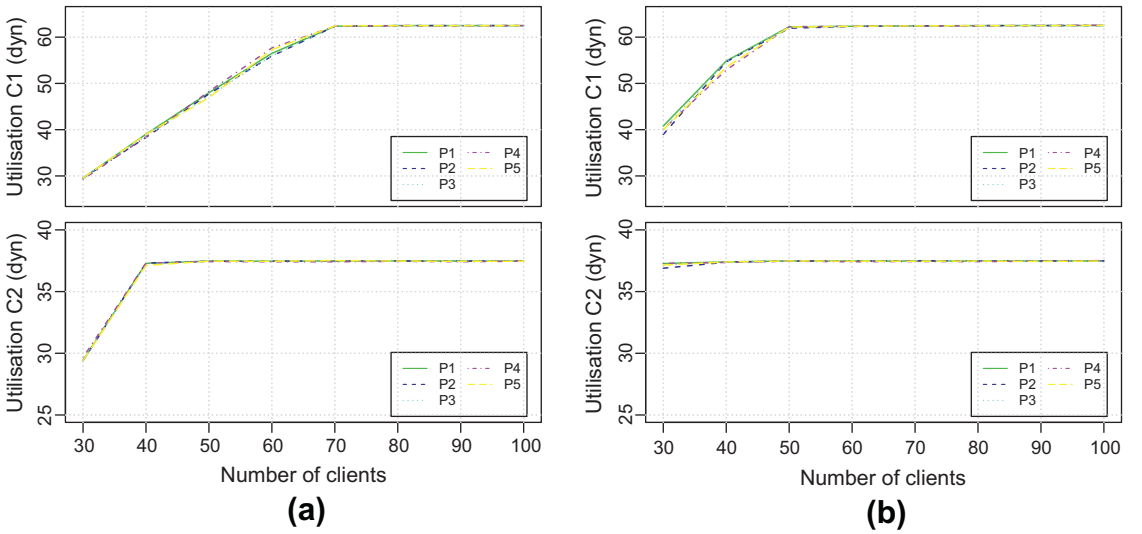

Fig. 7. 95th percentile of the average Application/Database server utilisation for dynamic requests: (a) Workload 1; (b) Workload 2.

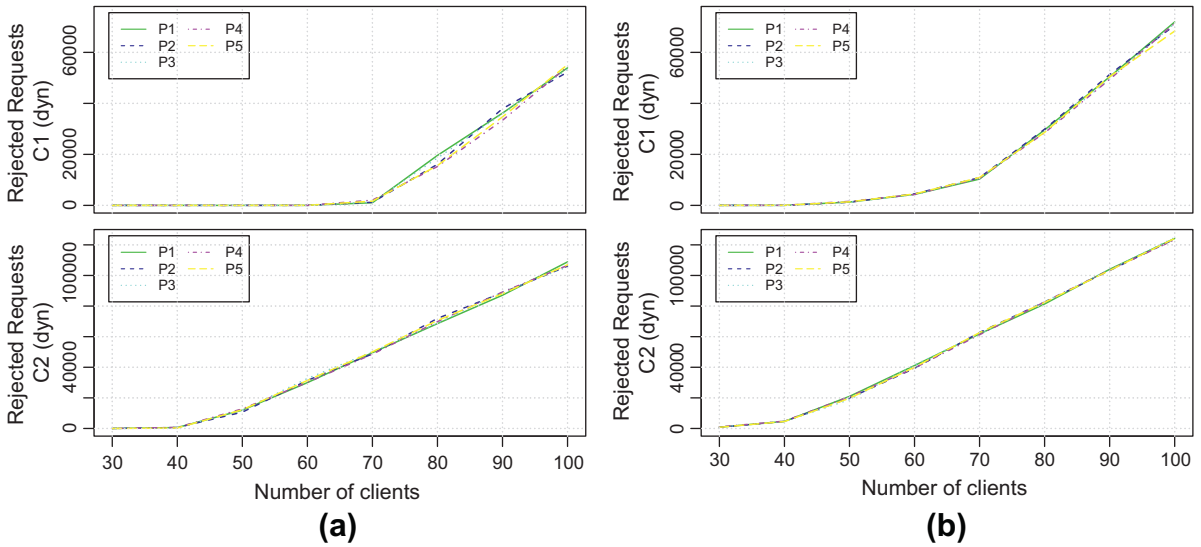

Fig. 8. Number of rejected dynamic requests: (a) Workload 1; (b) Workload 2.

algorithm achieves an extra goal, that is, the limitation of the response time regardless of the amount of traffic arriving to the system. The predictor that shows a good response time and the most stable behaviour is P4, as P5 shows some variability in 70, 80, 90 and 100 clients for *Workload 1*. We can also observe that there is not any differentiation in the response times obtained by class-1 and class-2 traffic, as we do not distinguish different queues in the Web and App/DB servers in order to keep the approach simple.

8.5. Downloaded pages

Sice the bottleneck of the Web system is the App/DB servers, we only represent the number of downloaded dynamic requests in Fig. 11. We can observe how, when the system receives the highest arrival rate (from 80 client to 100 clients), throughput predictors P4 and P5 download more class-1 dynamic Web pages compared to the rest of predictors with both workloads. We consider this more important than the fact that these predictors download less pages for 50, 60 and 70 clients, as the more traffic detected in the system, the more crucial the behaviour of the predictor is.

The response time and the number of downloaded dynamic Web pages obtained from the simulations lead us to the conclusion that P4 is the most suitable predictor for our admission control and load balancing algorithm. However, we would

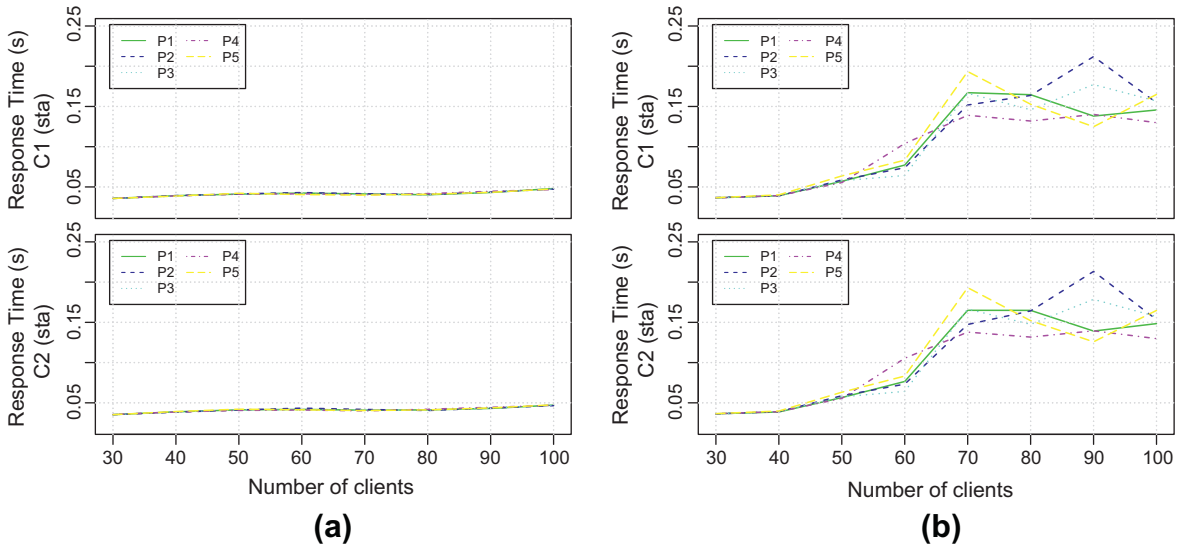

Fig. 9. 95th percentile of the response time for static requests: (a) Workload 1; (b) Workload 2.

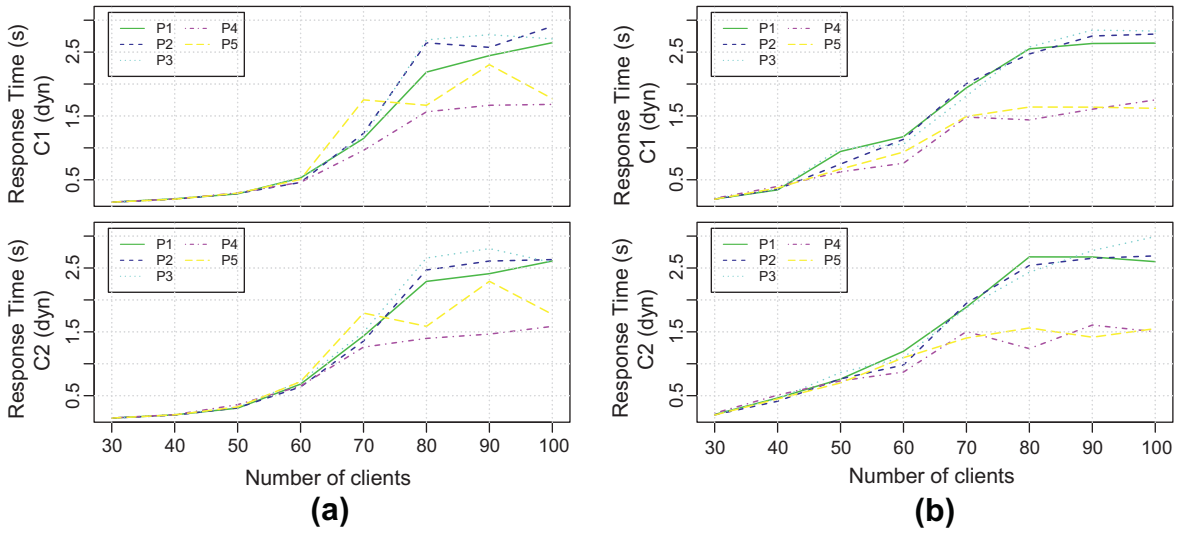

Fig. 10. 95th percentile of the response time for dynamic requests: (a) Workload 1; (b) Workload 2.

like to remark that the predictors P1, P2 and P3 do also obtain good performance results and that have an important advantage: they are easily obtained from the throughput of the two previous slots and that do not need a record of more previous slot throughput values as predictors P4 and P5, which are more complicated to compute.

9. Adaptive time slot scheduling compared to fixed time slot scheduling

In order to show the benefits of the adaptive time slot scheduling, we have configured our algorithm to be executed on a fixed time slot scheduling. The predictor chosen for these simulations is P3. We have redrawn some of previous figures, including the results obtained when invoking the algorithm periodically (named as "P3_per" in the figures). The workload chosen for this comparison is *Workload 2*.

9.1. Error in throughput predictions

Fig. 12 shows the mean square error of throughput predictions for both static and dynamic traffic. Very similar results are obtained for the adaptive time slot scheduling and the fixed time slot scheduling, so the prediction errors do not help in differentiating the scheduling policy.

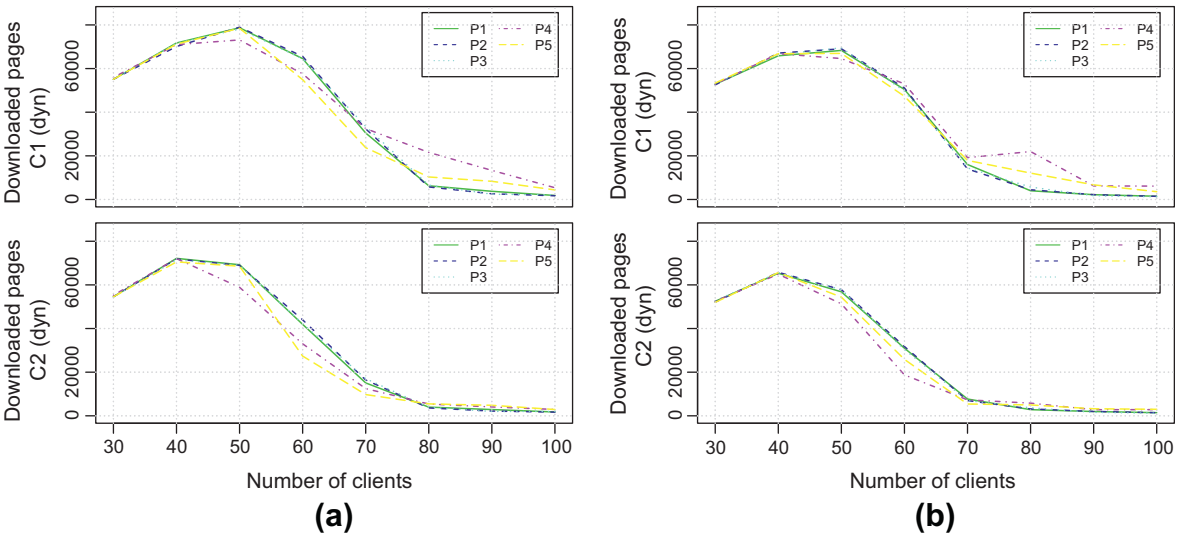

Fig. 11. Number of downloaded dynamic requests: (a) Workload 1; (b) Workload 2.

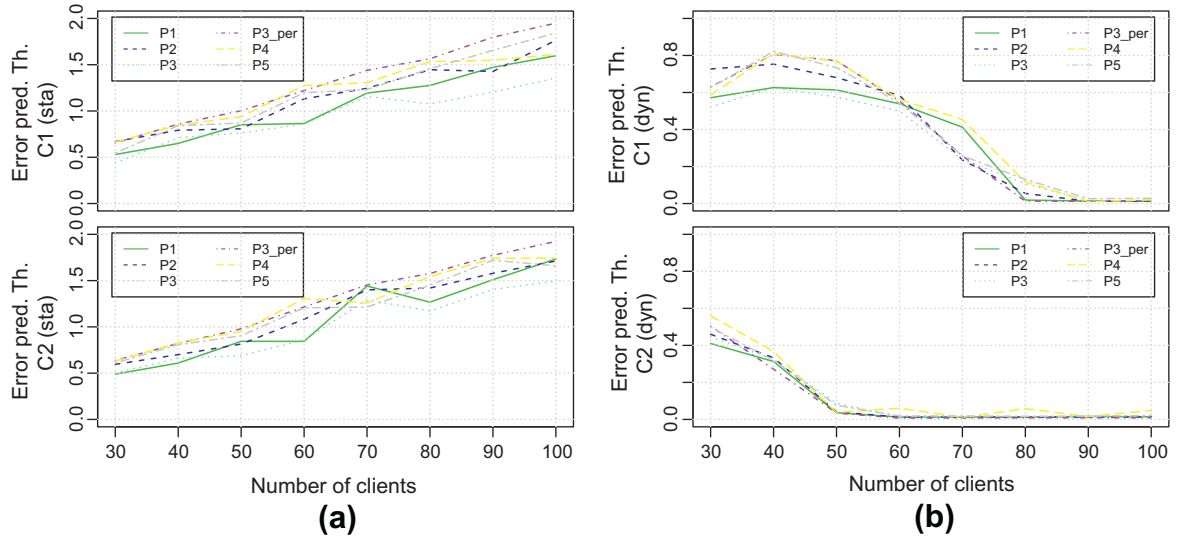

Fig. 12. Mean squared error of throughput predictions: (a) static requests; (b) dynamic requests.

9.2. Rejection of requests

Fig. 13 shows the number of rejected requests that ask for dynamic content. Once again, we cannot observe either a differentiation among the predictors, or in the scheduling.

9.3. CPU utilisation

The 95th percentile of the App/DB server utilisation is represented in Fig. 14. Here we observe that the utilisation level of the App/DB servers is lower for P3_per in the first points of the x-axis of the graph. In the case of class-1 traffic, the servers seem to be less loaded for 30, 40 and 50 clients with P3_per. The case of 30 clients also reaches a lower utilisation level for class-2 traffic.

However, if we analyse the P3_per utilisation level of class-2 traffic after 40 clients, we can also observe that it is slightly greater that the rest of the simulations. In fact, this indicates that the fixed time slot scheduling introduces some errors in the utilisation level reached for each traffic class. This also means that the algorithm is less accurate in its reservations and that the SLA is less guaranteed.

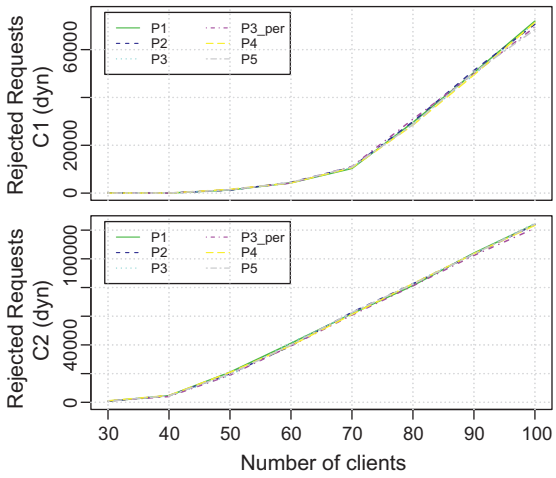

Fig. 13. Number of rejected dynamic requests.

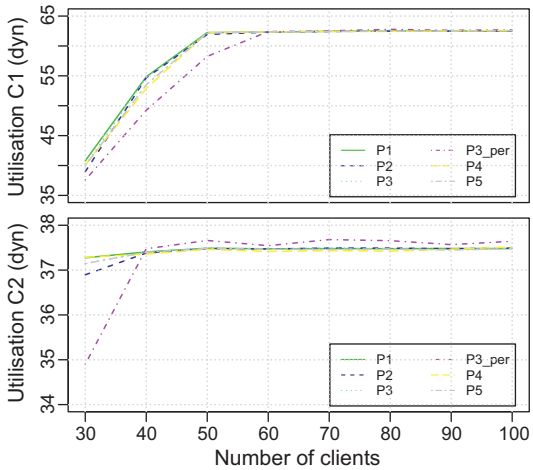

Fig. 14. 95th percentile of the App/DB server utilisation for dynamic requests.

9.4. Response time and downloaded requests

The response time and the number of downloaded requests gave us an idea of the predictor that suits our algorithm best in the previous section. Hence, we redraw these figures again in order to illustrate the behaviour of the P3 predictor under a periodical invocation scheme.

Fig. 15 shows the number of downloaded dynamic requests. It shows that P3_per downloads fewer class-1 requests than the rest of the simulation results. Also we can observe this same behaviour for class-2 traffic. Despite P3_per utilisation level for class-2 requests being greater than the rest of the simulations (as we have just seen in Fig. 14), it downloads fewer class-2 requests.

Finally, Fig. 16 shows the response time obtained by static and dynamic requests, respectively. The results obtained by the static requests do not give us much information, but the response time for dynamic request do. We can observe in Fig. 16b how P3_per increases its response time for most of the number of clients above the rest of the curves.

The results obtained for the response time and the number of downloaded dynamic requests confirm the fact that the simulation of the algorithm following a periodical invocation is outperformed by the adaptive invocation.

10. Comparison to IQRD

In Section 2, we reviewed the admission control mechanisms that exist in literature, and finally we have found a candidate proposal to be compared with our algorithm. It has been difficult to find because we needed the alternative solution to be QoS-aware and include an admission control and a content-aware load balancing strategy.

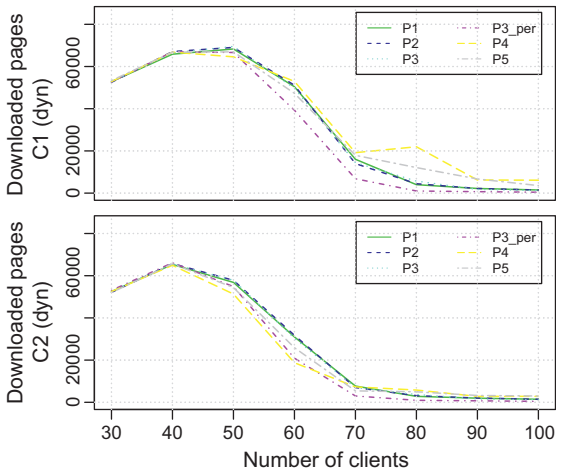

Fig. 15. Number of downloaded dynamic requests.

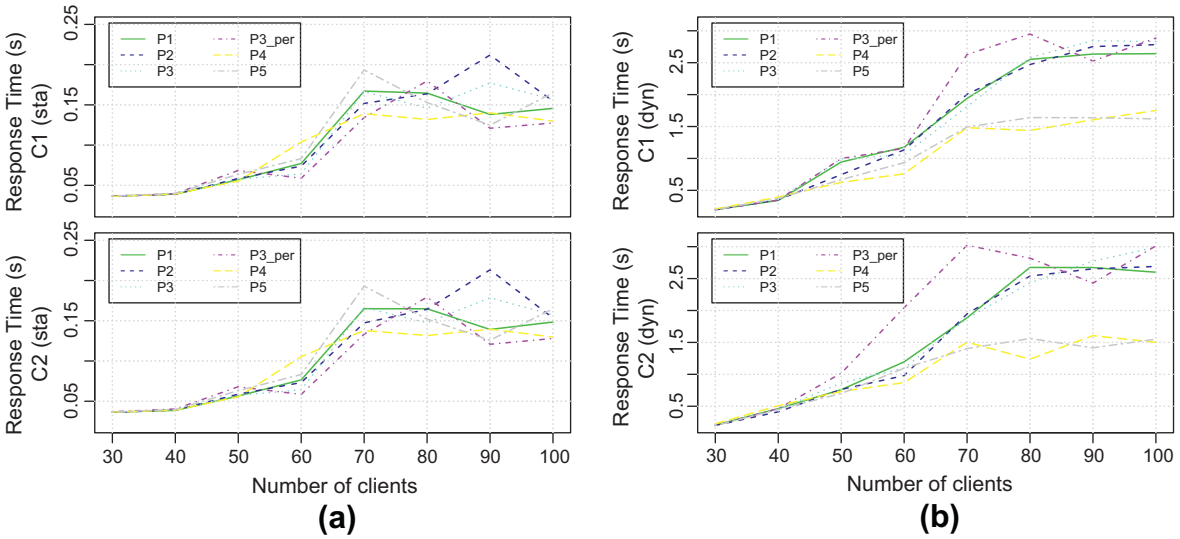

Fig. 16. 95th percentile of the response time: (a) static requests; (b) dynamic requests.

We decided to compare our algorithm with the IQRD, the solution described by Sharifian et al. in [41]. An important reason we have considered in selecting IQRD is because it divides the time into small intervals in order to avoid modelling errors. They consider a periodical algorithm invocation, and at the same time these periods (or slots) are divided into smaller intervals in order to obtain monitoring information from the Web servers. This is a completely different philosophy from ours and the results of this comparison may be quite interesting and lead us to make some conclusions about the possible reduction of overhead when invoking an algorithm in a Web environment.

The IQRD solution defines the SLA of the different services by the response time. Several Web servers are supposed to execute in a node in order to differentiate among the services offered. Each of the Web servers is modelled by a $M/P/1$ queueing network. Hence, the inter-arrival rate the IQRD model uses follows an exponential distribution. The response time guaranteed for each type of service permits it to obtain the maximum arrival rate that is going to be accepted during the next period. In order to obtain this value, the mean and the second moment of the service time are received from the Web servers. These metrics are monitored 100 times per period (or slot). The maximum arrival rate dictates the maximum number of requests of each service type which is going to be accepted during the next period.

In a given period, the maximum number of requests decreases by one in a Web server when a request of that class is accepted, and increases again when the request has been served. This would add an extra overhead in a one-way architecture as the server nodes should inform the distributor that a request has been served. The admission control starts rejecting requests when all the Web servers have the maximum number of requests equal to zero.

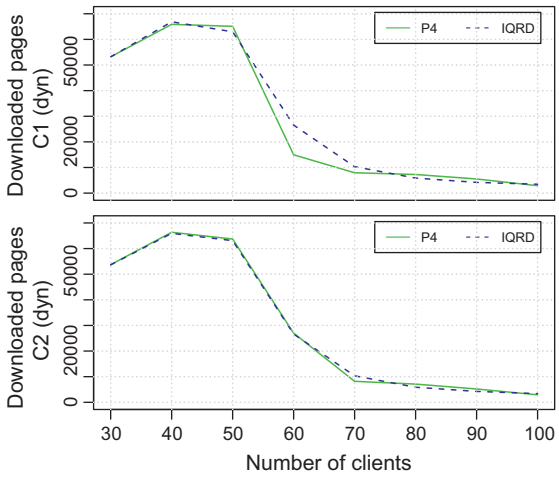

Fig. 17. Number of downloaded dynamic requests.

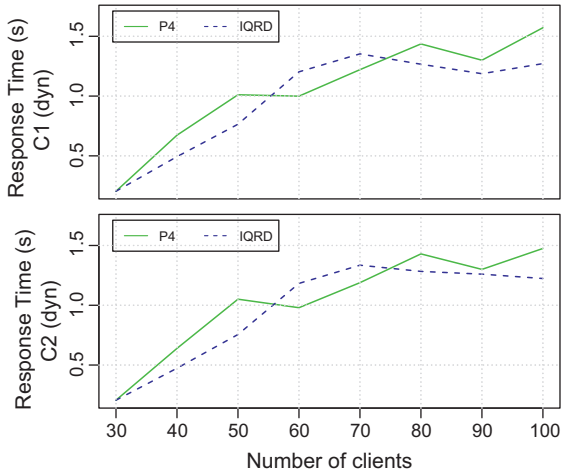

Fig. 18. 95th percentile of the response time for dynamic requests.

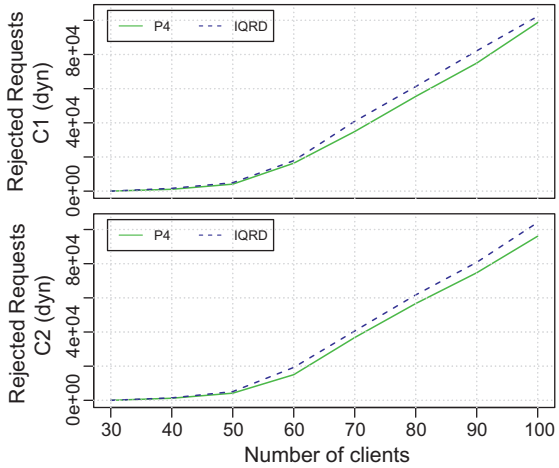

Fig. 19. Number of rejected dynamic requests.

In our implementation of OPNET Modeler, we have considered that only one Web server is executing in each Web and App/DB node. Hence, in order to simulate both algorithms under the same conditions, we have defined two different classes of service, c_1 and c_2, that in the case of IQRD have the same response time requirement, and in our case, have the same SLA value.

In order to represent our algorithm, we have chosen to simulate the predictor P4 with the SLA values $c_1 = c_2 = 0.5$. The workload used in both sets of simulations is *Workload 2*.

10.1. Response time and downloaded requests

We have represented the number of downloaded dynamic pages in Fig. 17. It can be observed that both curves are very similar, although IQRD downloads more class-1 requests for 60 and 70 clients than our algorithm, and our algorithm outperforms IQRD for 80 and 90 clients. However, both of them essentially download a similar number dynamic requests.

If we compare the plot of number of downloaded requests with Fig. 18, that shows the response time of dynamic requests for both algorithms, we can observe the two curves are very similar as well. The response time obtained by both classes is always below 1.6 s, despite IQRD achieving a better response time in the most extreme cases, that is with 80, 90 and 100 clients.

10.2. Rejection of requests

Fig. 19 depicts the number of dynamic rejected requests. In this figure it is clear that IQRD rejects more requests for most of the number of clients than our algorithm.

Therefore, we can conclude in this section that our algorithm obtains good performance results in comparison to IQRD, another admission control and content-aware load balancing algorithm that considers QoS. We have compared these two algorithms under the same circumstances and we have observed that both of them work well, despite our algorithm rejecting fewer requests when subjected to the same workload.

Several comments should be included as conclusions. In our aim to obtain low overhead, we do not consider a periodic invocation time in our algorithm as IQRD does. Indeed, the IQRD algorithm requires constantly fresh information from the servers, and subdivides the intervals 100 times to get information from them. This increases the overhead of the solution. However, there is also another detail in the IQRD proposal that will increase the overhead of the system, which is the fact that the algorithm needs to control the number of requests that can still be attended by the server: when they decrease, and more importantly, increase. When the service of a request is finished, the IQRD proposal requires an extra feedback from the Web servers as they have to indicate to the distributor module that the counter should be increased by one. Hence, we prove with these results that our algorithm obtains as good performance as that obtained by an algorithm that introduces more overhead than ours in the system.

11. Conclusions and open problems

We introduce a low overhead admission control and load balancing algorithm that bases its decisions on the values obtained by a throughput predictor. The invocation times of the algorithm are adaptively scheduled depending on the burstiness detected in the system in order to reduce the overhead of the algorithm. Five throughput predictors are introduced in this work and their behaviour in the admission control and load balancing algorithm is compared under a simulation scenario in OPNET Modeler.

The resource allocation algorithm adaptively distributes the utilisation of the servers among the different classes of requests. The results show that the algorithm guarantees the service specified in the SLA with all the throughput predictors. The response times of dynamic requests obtained by the algorithm and the number of downloaded dynamic requests lead us to the conclusion that the predictor that better suits to our algorithm is based on LMS. An extra goal is acquired by this algorithm: that is the limitation in the response time of the requests when it is congested. This means that service is guaranteed despite the arrival rate of requests reaching the Web system. In this paper, we also provide results that show the benefits of an adaptive time slot scheduling compared to a fixed time slot one. The comparison of the results obtained by our algorithm and IQRD shows us that our algorithm works reasonably well and its performance is very similar to the performance of IQRD while introducing less overhead.

Further research is needed to avoid an under-utilised model when the algorithm reserves resources for a forecasted traffic demand that does not arrive. This can lead to unnecessary rejections in lower priority classes. This problem may be solved by defining a probability index based on a record that stores the percentage index of arrival rate of requests for each class of service during some previous slots, which would estimate the chance of not receiving more priority requests and allow an increase in the utilisation margin of the lower priority requests. Future work also includes the use of optimisation techniques, combined with adaptive control in order to improve the resource allocation strategy we propose in this paper. Our algorithm is developed to be included in a locally distributed Web system, but could also form part of a wider geographically distributed load balancing architecture. This possibility has yet to be investigated further.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Education and Science under Grant TIN2006-02265. We are very grateful to the reviewers for their constructive comments that have helped us to improve this text.

References

[1] OPNET technologies, Inc., 2011, <http://www.opnet.com/>.
[2] The Internet Traffic Archive, 2011, <http://ita.ee.lbl.gov/>.
[3] T.F. Abdelzaher, K.G. Shin, N. Bhatti, Performance guarantees for web server end-systems: a control-theoretical approach, IEEE Transactions on Parallel and Distributed Systems 13 (2002) 80–96.
[4] J. Almeida, M. Dabu, P. Cao, Providing differentiated levels of service in web content hosting, in: Proc. of the First Workshop on Internet Server Performance, 1998.
[5] M. Andersson, J. Cao, M. Kihl, C. Nyberg, Admission control with service level agreements for a web server, in: Proc. of European Internet and Multimedia Systems and Applications, 2005.
[6] M. Arlitt, T. Jin, A workload characterization of the 1998 World Cup Web site, Technical report hpl-1999-35r1, HP Labs, October 1999.
[7] M. Aron, P. Druschel, W. Zwaenepoel, Efficient support for P-HTTP in cluster-based web servers, in: Proc. of the Annual Conference on USENIX Annual Technical Conference, 1999.
[8] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, Scalable content-aware request distribution in cluster-based network servers, in: Proc. of the USENIX 2000 Annual Technical Conference, 2000.
[9] J. Aweya, M. Ouellette, D.Y. Montuno, B. Doray, K. Felske, An adaptive load balancing scheme for web servers, International Journal of Network Management 12 (2002) 3–39.
[10] N. Bartolini, G. Bongiovanni, S. Silvestri, Self-* through self-learning: overload control for distributed web systems, Computer Networks 53 (2009) 727–743.
[11] N. Bhatti, R. Friedrich, Web server support for tiered services, IEEE Network September/October (1999) 64–71.
[12] H. Chen, P. Mohapatra, Overload control in QoS-aware web servers, Computer Networks 42 (2003) 119–133.
[13] X. Chen, H. Chen, P. Mohapatra, ACES: an efficient admission control scheme for QoS-aware web servers, Computer Communications 26 (2003) 1581–1593.
[14] S.-T. Cheng, C.-M. Chen, I.-R. Chen, Performance evaluation of an admission control algorithm: dynamic threshold with negotiation, Performance Evaluation 52 (2003) 1–13.
[15] L. Cherkasova, M. Karlsson, Scalable web server cluster design with workload-aware request distribution strategy WARD, in: Proc. of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS), 2001.
[16] L. Cherkasova, P. Phaal, Session-based admission control: a mechanism for peak load management of commercial web sites, IEEE Transactions on Computers 51 (2002) 669–685.
[17] L. Eggert, J. Heidemann, Application-level differentiated services for web servers, World Wide Web 2 (1999) 133–142.
[18] S. Elnikety, E. Nahum, J. Tracey, W. Zwaenepoel, A method for transparent admission control and request scheduling in e-commerce web sites, in: Proc. of the 13th International Conference on World Wide Web, 2004.
[19] R.G. Garroppo, S. Giordano, M. Pagano, G. Procissi, On traffic prediction for resource allocation: a chebyshev bound based allocation scheme, Computer Communications 31 (2008) 3741–3751.
[20] K. Gilly, S. Alcaraz, C. Juiz, R. Puigjaner, Comparison of predictive techniques in cluster-based network servers with resource allocation, in: Proc. of the 12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2004.
[21] K. Gilly, S. Alcaraz, C. Juiz, R. Puigjaner, Analysis of burstiness monitoring and detection in an adaptive web system, Computer Networks 53 (2009) 668–679.
[22] K. Gilly, C. Juiz, S. Alcaraz, R. Puigjaner, Adaptive admission control algorithm in a QoS-aware web system, in: Proc. of IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2009.
[23] G.C. Goodwin, K.S. Sin, Adaptive Filtering: Prediction and Control, Prentice-Hall, 1984.
[24] S. Haykin, Adaptive Filter Theory, Prentice-Hall, 1991.
[25] Q. He, J. Yan, R. Kowalczyk, H. Jin, Y. Yang, Lifetime service level agreement management with autonomous agents for services provision, Information Sciences 179 (2009) 2591–2605.
[26] G. Hunt, E. Nahum, J. Tracey, Enabling content-based load distribution for scalable services, Tech. rep., IBM T.J. Watson Research Center, May 1997.
[27] V. Kanodia, E.W. Knightly, Multi-class latency-bounded web services, in: Proc. of the 8th International Workshop onQuality of Service (IWQOS), 2000.
[28] D. Kerdlapanan, A. Khunkitti, Content-based load balancing with multicast and tcp-handoff, in: Proc. of International Symposium on Circuits and Systems, 2003.
[29] M. Kihl, A. Robertsson, M. Andersson, B. Wittenmark, Control-theoretic analysis of admission control mechanisms for web server systems, World Wide Web 11 (2008) 93–116.
[30] M. Kihl, N. Widell, Admission control schemes guaranteeing customer QOS in commercial web sites, in: Proc. of the IFIP TC6/WG6.2 & WG6.7 Conference on Network Control and Engineering for QoS, Security and Mobility, 2002.
[31] Y.-D. Lin, P.-T. Tsai, P.-C. Lin, C.-M. Tien, Direct web switch routing with state migration, TCP masquerade, and cookie name rewriting, in: Proc. of Global Telecommunications Conference, 2003.
[32] H.-H. Liu, M.-L. Chiang, M.-C. Wu, Efficient support for content-aware request distribution and persistent connection in Web clusters, Software: Practise and Experience 37 (2007) 1215–1241.
[33] M.-Y. Luo, C.-S. Yang, C.-W. Tseng, Analysis and improvement of content-aware routing mechanisms, IEICE Transactions on Communications E88 (2005) 227–238.
[34] D.T. Mcwherter, B. Schroeder, A. Ailamaki, M. Harchol-balter, Priority mechanisms for oltp and transactional web applications, in: Proc. of the 20th International Conference on Data Engineering, 2004.
[35] J. Oh, N.W. Cho, H. Kim, Y. Min, S.-H. Kang, Dynamic execution planning for reliable collaborative business processes, Information Sciences 181 (2011) 351–361.
[36] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E.M. Nahum, Locality-aware request distribution in cluster-based network servers, in: Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS), 1998.
[37] R. Pandey, J. Fritz, B.R. Olsson, Supporting quality of service in http servers, in: Proc. of the 7th Annual SIGACT-SIGOPS Symposium on Principles of Distributed Computing, 1998.
[38] A.E. Papathanasiou, E.V. Hensbergen, KNITS: switch-based connection hand-off, in: Proc. of IEEE INFOCOM, 2002.
[39] N. Poggi, T. Moreno, J.L. Berral, R. Gavald, J. Torres, Self-adaptive utility-based web session management, Computer Networks 53 (2009) 1712–1721.
[40] B. Schroeder, M. Harchol-Balter, Web servers under overload: how scheduling can help, ACM Transactions on Internet Technology 6 (2006) 20–52.
[41] S. Sharifian, S.A. Motamedi, M.K. Akbarib, A content-based load balancing algorithm with admission control for cluster web servers, Future Generation Computer Systems 24 (2008) 775–787.

[42] Y.-F. Sit, C.-L. Wang, F. Lau, Cyclone: a high-performance cluster-based web server with socket cloning, Cluster Computing 7 (2004) 21–37.
[43] B. Urgaonkar, P. Shenoy, Cataclysm: scalable overload policing for internet applications, Journal of Network and Computer Applications 31 (2008) 891–920.
[44] T. Voigt, P. Gunningberg, Adaptive resource-based web server admission control, in: Proc. of the 7th International Symposium on Computers and Communications, 2002.
[45] M. Welsh, D. Culler, Adaptive overload control for busy internet servers, in: Proc. of the 4th conference on USENIX Symposium on Internet Technologies and Systems, 2003.
[46] B. Widrow, J. John.R. Glover, J.M. McCool, J. Kaunitz, C.S. Williams, R.H. Hearn, J.R. Zeidler, J. Eugene Dong, R.C. Goodlin, Adaptive noise cancelling: principles and applications, Proceedings of the IEEE 63 (1975) 1692–1716.
[47] D. Zeng-Kai, J. Jiu-Bin, A completely distributed architecture for cluster-based web servers, in: Proc. of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2003.
[48] J. Zhou, T. Yang, Selective early request termination for busy internet services, in: Proc. of the 15th International Conference on World Wide Web, 2006.