A Priority-based Admission Control Scheme for Commercial Web Servers

Ibtehal Nafea [1]. Muhammad Younas . Robert Holton . Irfan Awan

Abstract: This paper investigates into the performance and load management of web servers that are deployed in commercial websites. Such websites offer various services such as flight/hotel booking, online banking, stock trading, and product purchases among others. Customers are increasingly relying on these round-the-clock services which are easier and (generally) cheaper to order. However, such an increasing number of customers' requests makes a greater demand on the web servers. This leads to web servers' overload and the consequential provisioning of inadequate level of service. This paper addresses these issues and proposes an admission control scheme which is based on the class-based priority scheme that classifies customer's requests into different classes. The proposed scheme is formally specified using π-calculus and is implemented as a Java-based prototype system. The prototype system is used to simulate the behaviour of commercial website servers and to evaluate their performance in terms of response time, throughput, arrival rate, and the percentage of dropped requests. Experimental results demonstrate that the proposed scheme significantly improves the performance of high priority requests but without causing adverse effects on low priority requests.

Keywords: web servers, performance, efficiency, web portals, commercial websites

1. Introduction

The web has undergone substantial changes from serving simple static web pages to a medium for conducting online business, marketing, and financial transactions, and for offering a variety of other commercial services. This dramatic change in the online service offering has also caused a rapid rise in the number of users using the web. Such an increasing population of users generally leads to overload conditions, low quality of service and poor performance by commercial servers. However, it is extremely undesirable for a commercial website to provide inadequate quality of service as this would have negative effects on the image of business. For example, if a web site takes one minute to load, it is quite possible that the user will leave that site for an alternative one.

This paper focuses on an extremely important and challenging research issue of web server performance and overload problems in the context of commercial websites

[1] Ibtehal Nafea,
College of Computer Science and Engineering
Taibah University, Saudi Arabia
e-mail: e.t.nafea@gmail.com

Muhammad Younas
Department of Computing and Communication Technologies
Oxford Brookes University, Oxford, UK
e-mail: m.younas@brookes.ac.uk

Rob Holton, Irfan Awan
University of Bradford, Bradford, UK
email: i.u.awan@bradford.ac.uk
email: d.r.w.holton@Bradford.ac.uk

that serve large population of users. The problems of web server performance are not new and several solutions have been proposed to alleviate such problems. These include, for example, (i) clusters of multiple web servers which improve response time and minimise server overload [3] (ii) cache servers which are used to improve the performance of web servers through caching information [4], and (iii) other request scheduling mechanisms have been proposed to improve the performance of web servers [5, 6, 7]. For instance, dynamic resource management can help provide improved performance, maintain acceptable response time and minimise server overload by allocating resources to other service providers and reduces service degradation.

This paper aims to synthesize the performance management mechanisms of admission control, service differentiation and degradation, and requests scheduling in a systematic manner in order to address performance issues of commercial website servers by taking into account commercial web portals (or price comparison websites), classification of requests, recommender (or review) websites and the effects of dropped (rejected) requests. The practice of employing individual techniques such as admission control [5] or request scheduling [19] is inappropriate for modern commercial servers which operate in a more complex scenario than classical web servers that serve simple static web pages. Figure 1 presents a generalized architecture of the modern commercial web servers' setup. Web servers (provider or seller web server) can receive requests from commercial web portals or directly from clients. For instance, clients can directly go to the British Airways (BA) website, check the flights and make booking. Alternatively, they can first go to a price comparison website (to compare the BA prices with other airlines) or a recommender website (to read user's reviews on BA quality) and then go the BA website to make a booking.

In the following, we give a rationale as to why different requests need different level of service depending on their priority, the originating sources and the capacity of web servers.

- *Types of requests*: Users interact with websites by submitting various types of requests. For example, searching or browsing a web site for hotel prices, reading/writing products reviews and recommendation, selecting/adding products to shopping cart, and making payment. To the service provider (online seller) some of these requests can be more important than others. For instance, processing payment and add-to-cart requests can be more important than search or browse. An analysis of the literature has revealed that the number of search and browse requests is significantly higher than payment requests. According to [1], the percentage of customers who buy items is significantly lower than those who usually use commercial websites to find information such as air fares or book prices, but without buying anything. Similarly, other research studies [2] report that the number of users (who buy items from the Internet) is 5% (see [2] for details). In addition, other studies reveal that 89 % of users search for products on different websites before making purchases [24]. Similarly, 65% of hotel buyers check three sites before actually booking a room [25]. Further, more than two thirds of UK shoppers are reported to cancel orders even if they have added products to their online shopping carts [26].
  The large number of search and browse requests has performance consequences, as they severely affect the processing and response time of important requests, such as payment or add to cart requests.
- *Sources of requests*: Requests can be originated or generated from different sources. Users can directly submit requests through websites or they may come from web portals such as price comparison websites (e.g., travelsupermarket.com)

or from recommender websites (www.trustedreviews.com/ or Which?<www.which.co.uk/) as these days it is common practice that customers review products before buying them. Requests originating from web portals, reviews websites or recommender websites and submitted to seller web servers are more likely to be converted to 'buy' requests as those users would have already done a market analysis.

- *Web server overload*: Several factors contribute to the web server overload. For example, high volume of traffic can cause server to slow down and/or reject requests. Server may process too many database queries or it may have minimal memory to handle the processes being run – e.g., commercial websites frequently offer promotional sales that significantly increase traffic. Servers may also process large number of simultaneous requests downloading large files (e.g., images or videos of products). All such events result in overload that could be an order of magnitude higher than normal load. Overload conditions have adverse effects on the service provisioning of web servers and may result in lost revenue. Thus appropriate schemes should be developed in order to optimise the performance of web servers. One of the options is to minimise the server processing time spent on requests which are eventually not serviced due to overload conditions.

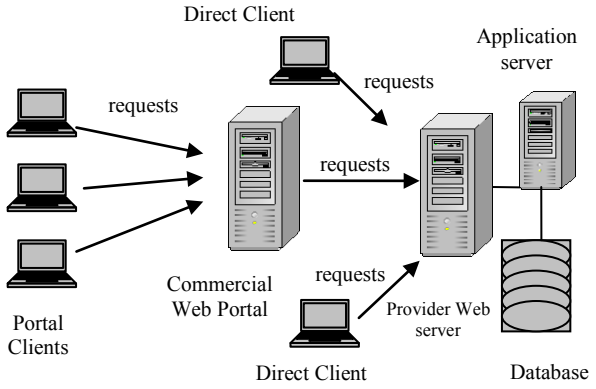

Figure 1. Architecture of Commercial Web Servers

Taking account of the above factors, we propose a priority-based admission control scheme that classifies requests into different classes. We contend that by assigning class-based priorities at multiple service levels, web servers can perform better and can improve the performance of more important requests without causing adverse effects on low priority requests. The proposed scheme is formally specified using π-calculus and is implemented as a Java-based prototype system. The prototype system is used to simulate the behaviour of commercial website servers and to evaluate their performance in terms of response time, arrival rate, and the percentage of dropped requests.

The remainder of the paper is structured as follows. Section 2 gives a review of related work. Section 3 presents the architectural aspects of the proposed scheme. Formal specification of the proposed scheme is described in section 4. The prototype implementation is presented in section 5. Section 6 describes experimental results and their analysis. Section 7 concludes the paper.

2. Related Work

Many techniques have been developed in order to improve the performance of commercial services, including: request scheduling, admission control, service differentiation, dynamic resource management, and service degradation. It is widely accepted that the web server's behaviour has a strong influence on the commercial services. For example, if the server becomes overloaded the response time can grow to an unacceptable level that can lead to the user leaving the website. To maintain acceptable response time and minimise server overload, clusters of multiple web servers have been developed [2, 12]. Cache servers also help to improve the performance of web servers [4], by reducing the response time in real-world dynamic web applications.

Mechanisms to schedule requests have been proposed to improve the performance of web servers [5, 6, 7]. Request scheduling refers to the order in which concurrent requests should be served. Some techniques modify the scheduler process in a web server to decide the order in which the requests should be handled so as to improve the mean response time. The main idea underlying these techniques is to classify the requests and schedule them in a given order, as a result of providing different quality of service (QoS) levels to each group, by assigning different priorities to the different requests. For example, the *shortest remaining processing time first* (SRPT) technique prioritises the service of shortest requests (for smaller size static web content) over long requests (for large size web content) [9, 10]. SRPT scheduling is claimed to provide a better response time to short requests at relatively low costs to the long requests. However, Crovella et al. [9] indicate that the application level scheduling does not provide fine enough control over the order in which packets enter the network. The other problem is that in traditional UNIX network stack implementations, processing for all connections is handled in an aggregate manner. That is, outgoing packets are placed on the wire in response to the arrival of acknowledgements. This means that if many connections have data ready to send, and if the client and network are not the bottleneck, then data will be sent from the set of connections and acknowledgements will arrive, which is not under application control.

Yue et al. [11] present a profit-aware admission control mechanism for overload protection in E-commerce websites. This approach classifies clients into two categories: premium customers (with previous purchase records) and basic customers (having no purchase records). Priority is given to the requests of premium customers on the basis that these customers are more likely to make purchases whenever they visit the website. This approach also employs hashing tables with full IP address and network ID prefix, in order to maintain records of the purchases of clients in a fine-grain and coarse-grain manner. However, this approach is not realistic due to recent dynamic IP address technology that allocates new IP address each time for customers. Changwoo et al. [23] study the delay distribution of a generalization of the class of scheduling policies called SMART, which includes SRPT, PSJF, and a range of practical variants, in a discrete-time queueing system.

Bhatti et al. [16] propose that the architecture of web servers can provide QoS to differentiated clients. They used request classification, admission control and request scheduling to support distinct performance levels for different classes of clients. They classified the requests according to the clients' preference. The admission control of low priority requests was then triggered when thresholds in the number of requests were queued and the number of premium requests queued were exceeded. The proposed architecture considers only static web content. Thus, they are not directly applicable to multi-tiered sites or the dynamic web content as in commercial web servers. Verma et al. [17] propose a service time-based online admission control

methodology for maximising the profits of a service provider. The admission control of requests uses the shortest remaining job first (SRJF) policy which is restricted to a set of undecided requests, i.e. the requests which have neither been rejected nor serviced. Admission control rejects some of the requests that may maximise the profit of the service provider, so the remaining requests can be serviced within their QoS bound.

Further, a number of studies have focused on managing sessions to prevent overload in session-based applications. Chen et al. [18] illustrate a commercial web server log analyser for deriving session-based dependency relationships among HTTP requests. They proposed a dynamic weighted fair sharing (DWFS) scheduling algorithm to control overload. DWFS is claimed to improve server response time. Muppala et al. [20] propose session-based admission control approaches for multi-tier Internet applications, including, measurement based admission control (MBAC) and coordinated session-based admission control approach (CoSAC). These approaches aim to improve the effective session throughput.

Almeida et al. [13] propose joint resource allocation and admission control solutions that are designed to consider the provider's revenues, the cost of resource utilisation and the customers' QoS requirements which are specified in terms of the response time of the individual requests. They resolved the optimisation problem by means of an analytical queuing-based solution. Urgaonkar et al. [15] implemented the QoS adaptive degradation approach. They considered that during overload conditions the performance of the admitted requests can be degraded within the limits established by the service level agreement (SLA). Almeida et al. [22] propose an optimisation model that identifies the optimal resource allocation by maximising a provider's revenues while satisfying customers QoS constraints and minimising resource usage cost. They describe two tightly inter-related problems in autonomic computing, namely, a short-term resource allocation problem and a long-term capacity planning problem; they use queuing models to address the resource management problems in autonomic service-oriented architectures.

Control theory has been generally used to modify the behaviour of dynamic systems. Several works make use of control theory to: avoid overload, meet the individual response time and to guarantee throughput. Abdelzaher et al. [21] describe performance control of a web server using classical feedback control theory. The authors use feedback control theory to achieve overload protection, performance guarantees and service differentiation, in the presence of load unpredictability.

3. The Proposed Scheme

This section gives an overview of the proposed Priority-based Admission Control (PAC) scheme and illustrates its main elements. The proposed scheme is based on our previous work [14], but is extended in various directions in order to cater for commercial web portals, distinct types of assigning priorities, and the effects of high priority requests on low priority requests.

The proposed PAC scheme is to synthesize the performance management mechanisms of admission control, requests scheduling and service differentiation mechanisms, in order to develop new criteria for effectively managing the performance of modern commercial web servers which are generally provisioned through an integrated architecture as depicted in Figure 2. In the commercial websites, users interact with web servers through a series of requests in order to acquire required information or make purchases. Example of such requests include: search for particular products, browse information about a specific item, browse/read product reviews and recommendation, select items, add items to cart, and make payment. These requests have different types and they originate from different

sources. They therefore need different policies for admission into the web servers, different priorities for scheduling, and different level of service.

Admission control: The purpose of admission control is to prevent web servers from entering into overload conditions. Such conditions occur due to requests coming from large numbers of clients. The admission control system tells clients that their requests will be accepted, served in a certain amount of time, or rejected immediately. For instance, the server, without sufficient capacity, will reject low priority requests when there are high priority requests in the system. Different algorithms are implemented, as part of the admission control strategy, which process requests depending on their type (high or low priority as stated below) or the status of the underlying servers. In the proposed scheme, admission control is implemented as a set of finite capacity virtual buffers that accommodate incoming requests. The proposed scheme also supports session-based admission of requests. Sessions are created for each client, which include different types of request, such as: browse, review, portal buy and direct buy. Session's requests are serviced only if the web server has a sufficient capacity.

Service differentiation and scheduling: The proposed scheme classifies requests into different classes. Requests can take form of browse, search, add to cart, buy, pay and review requests. Such requests are differentiated depending on the type of requests or the originating sources of the requests, for example, requests coming directly from clients or from web portal or a recommender system. The proposed scheme exploits priority scheduling mechanism that distinguishes classes of requests and schedules these classes in the order of priority. Depending on the type of requests, the scheduler assigns appropriate priorities. For example, buy or payment requests are prioritised over browse or search requests. Scheduling priority mechanisms, such as head-of-line (HoL) and pre-emptive resume (PR) take into account the situation that some requests may tolerate longer delays than others. In HoL, a request with highest priority upon its arrival always gets place in front of the low priority requests in the queue. It only waits either for the remaining service time of the low priority request being processed or the sum of service times of all the high priority requests in front of it. In PR, the high priority request pre-empts the low priority request being processed and can start its processing immediately provided there is no high priority request in front of it. The pre-empted request resumes its processing soon after the high priority requests is finished. The proposed scheme exploits the HoL mechanism in the request scheduling.

In order to model the proposed PAC scheme we employ queuing modelling technique. Separate virtual buffers for each class are maintained to temporarily hold each type of request. Instead of a single n-place buffer, the approach uses 'n' active components, each storing one class of requests such as high priority and low priority requests. Requests are then processed according to their priority and the availability of server's capacity. For example, if the buffer of high priority requests is not full then the system will first process the high priority request (e.g., add-to-cart or payment). But if the buffer of high priority requests is empty then the system will process the low priority request (e.g., browse or search) which is the first in the buffer of low priority request. Within each buffer, requests (of the same class) are scheduled according to first-in, first-out (FIFO) order.

Figure 2 depicts the main components of the proposed PAC scheme, which are explained as follows:

Gatekeeper (GK): This component deals with admission control. It also assigns a new Handler to each new client. Further details on GK are provided in the next section on formal specification of the proposed scheme.

Handler: Each arriving request is assigned to a Handler. Handler then passes links from each client to the appropriate virtual buffer for classification and processing.

Scheduler: The scheduler deals with the thread priorities that are created for each client. It schedules requests (or threads) so that they can be processed by a processor (or server).

Counter: It keeps a record of the number of each type of request that is currently handled.

Provider web server: This is the web server on the provider or seller side. It receives requests and processes them using application servers and database system. In the proposed scheme we simulate web server as a Processor component (see formal specification).

Client: This component represents client side of the system. Clients can be Direct Clients which directly submit requests to the provider web server or they can be Portal Clients which first submit requests to portal web server and then to the provider web server.

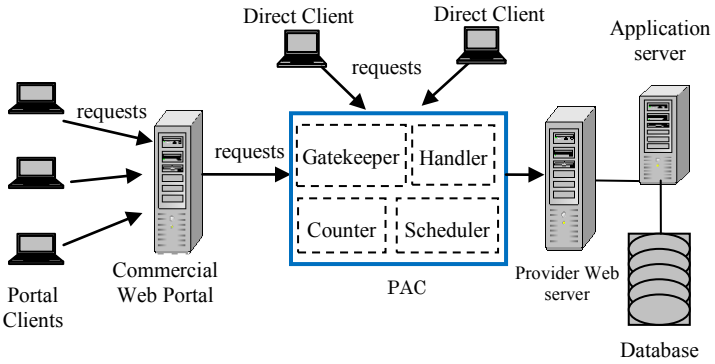

Figure 2. Architecture of the Proposed Scheme

4. Formal Specification

We design the proposed approach using the formal specification language of π-calculus [8]. We use π-calculus because the client requests can be expressed compactly. Also the compositional nature of the π-calculus allows the substitution of one component (e.g., client or server) for another. In addition, formal specification process allows us to rapidly investigate a number of different protocols for scheduling commercial web servers. Our models facilitate any behaviour construction such as changing priorities and adding client histories. We believe that our method is appropriate to modelling the complex architecture of the modern commercial web servers. The main notations used in the formal specification of the PAC scheme are described in Table1:

Table 1: Notations of π-calculus

Symbol	Description
$p\|q$	parallel composition
$\bar{x}\langle y\rangle$	To send y along x
$x(y)$	To receive y along x
$(new\ x)P$	New communication scope
$A(\vec{a}) \stackrel{def}{=} P_A$	Process definition
p + q	Choice operator
if ….then	Condition statement
$\vec{a}$	A vector of names
$p.q$	sequencing

Using the notation in Table 1, we formally specify the main components of the proposed PAC scheme shown in Figure 2. These include, Gatekeeper (GK), Handler, Scheduler, Counter, and Client.

The Overall System:

The overall system of a commercial website includes various components such as clients, web servers, portal servers, application servers and data stores. Here, we are interested in the formal specification of the two main components, client and server. Client component represents all the clients who are submitting requests to the commercial websites. Server component represents the main components of the proposed PAC scheme.

In the following notation we use eCommerce to represent the two main components (Client and Server) of a commercial website which are of particular interest to the proposed scheme. Thus using π-calculus, eCommerce, can be formally specified as:

$$\text{eCommerce} \stackrel{def}{=} \text{new connect, Client} \mid \text{Client} \mid \ldots \mid \text{Client} \mid \text{Server}$$

This indicates that the Server publishes a single action ′connect′ used by all clients to request service from a web server.

4.1 The Baseline Model

This section presents the formal specification of the base line model. In section 4.2, we extend the baseline model to include requests from review or recommender websites and also requests from commercial web portals.

Client:

Each client is associated with a Handler as described above. The client receives a link from the Server which is then used to connect it with a dedicated Handler. The client then submits various requests to the system via Handler. After the client has submitted ‘n’ number of low/high priority requests (e.g, browse, search, add-to-cart) requests, it informs the Handler that it has finished its requests. It then terminates the session.

The Client is specified as follows:

$$\text{Client} \stackrel{def}{=} \text{connect(link)}.\ \text{Client}'_0\ \langle \text{link} \rangle$$

$$\text{Client`}_i(\text{link}) \stackrel{\text{def}}{=} \overline{\text{link}}\langle\text{buy}\rangle.\,\text{link}(\text{resp}).\,\text{Client`}_{i+1}\langle\text{link}\rangle \qquad 0 \le i < n$$

$$+\overline{\text{link}}\langle\text{browse}\rangle.\,\text{link}(\text{resp}).\,\text{Client`}_{i+1}\langle\text{link}\rangle$$

$$\text{Client`}_n(\text{link}) \stackrel{\text{def}}{=} \overline{\text{link}}\langle\text{end}\rangle.\,0 \qquad n \in N$$

The specification shows the client making a non-deterministic choice between browse and buy requests; this is right since the aim of this specification is to describe how the component may interact, not why a particular action occurs. The implementation replaces this non-deterministic choice with one based on the relative probabilities of the two actions.

The Server:

The Server is made up of a number of components, including: Gatekeeper, Handler, Processor and a Counter. It is specified as follow.

$$\text{Server} \stackrel{\text{def}}{=} \text{new bind}, \overrightarrow{\text{up}}, \overrightarrow{\text{down}}$$
$$\text{Gatekeeper}|\text{Processor}|\text{Counter}_0\langle \text{up}_1, \text{down}_1\rangle \;|\text{Counter}_0\,\langle \text{up}_2, \text{down}_2\rangle$$

This specification indicates that the Server publishes information that is used by the counters, each of which uses a distinguished member from the vector $\overrightarrow{\text{up}}, \overrightarrow{\text{down}}$. The bound action connect is restricted to:

- Gatekeeper that passes action for client to handler
- Processor that processes the request and send the response and
- Counter that checks action to submit the request from handler to processor.

Gatekeeper:

When a client connects to the gatekeeper, a fresh action is passed to both the client and a new instance of the Handler, allowing these components to interact privately.

$$\text{Gatekeeper} \stackrel{\text{def}}{=} \text{new link } \overline{\text{connect}}\langle\text{link}\rangle.\,\text{Gatekeeper}|\text{Handler}\,\langle\text{link}\rangle$$

Handler:

Firstly, the handler receives a request from its client. If the request indicates that the client has finished, then the handler terminates; otherwise, the handler passes links to the appropriate counter of the handler's component, which checks to see whether there are too many requests in the system that need to be processed. If yes, the client is informed that the request has been rejected; otherwise, the request is processed, the counter is decremented, and the results of the query passed to the client. The handler then waits for the next request to be processed.

$$\text{Handler}(\text{link}) \stackrel{\text{def}}{=} \text{link}(\text{req})$$
$$($$
$$\text{if req} = \text{end then } 0$$
$$+$$
$$\text{if req} = \text{buy then Handler`}\langle \text{link}, \text{req}, \text{up}_1, \text{down}_1\rangle$$
$$+$$

$$\text{if req} = \text{browse then Handler`}\langle \text{link}, \text{req}, \text{up}_2, \text{down}_2 \rangle$$
$$)$$

$$\text{Handler`}(\text{link}, \text{req}, \text{full}, \text{dec}, \text{connect}) \stackrel{\text{def}}{\equiv} \text{full}(\text{ans})$$

$$($$

$$\text{if ans} = \text{yes then } \overline{\text{link}}\langle \text{reject} \rangle.\text{Handler}\langle \text{link} \rangle$$

$$+$$

$$\text{if ans} = \text{no then Process}\langle \text{link}, \text{req}, \text{dec} \rangle$$

$$)$$

$$\text{Process}(\text{link}, \text{req}, \text{dec}) \stackrel{\text{def}}{\equiv} \overline{\text{connect}}\langle \text{req} \rangle.\text{connect}(\text{resp}).\overline{\text{dec}}.\overline{\text{link}}\langle \text{resp} \rangle.\ \text{Handler}\langle \text{link} \rangle$$

Processor:

This represents the provider (or seller) web server as shown in Figure 2 above. It receives a request from a client via handler. After receiving it process the requests and then returns the output of the results to client. The Processor models the shared resources of a commercial website that need to be accessed under mutual exclusion. The proposed model has only one processor initially, but the model could be expanded to include several processors in order to represent a website with multiple web servers.

$$\text{Processor} \stackrel{\text{def}}{\equiv} \text{connect}(\text{req}).\overline{\text{connect}}\langle \text{resp} \rangle.\ \text{Processor}$$

Counter:

The counter keeps track of how many active requests of a particular type (buy, browse, etc.) there are in the system. When the system reaches its maximum capacity, the counter reports to the handler that no more requests can be accepted. Note that this approach is implemented using finite storage capabilities. This is to eliminate the starvation of low priority requests. For instance, if a high priority virtual buffer does not become empty, then a low priority request will never be processed.

$$\text{Counter}_0(\text{up}, \text{down}) \stackrel{\text{def}}{\equiv} \overline{\text{up}}\langle \text{no} \rangle.\text{Counter}_1\langle \text{up}, \text{down} \rangle$$

$$\text{Counter}_i(\text{up}, \text{down}) \stackrel{\text{def}}{\equiv} \overline{\text{up}}\langle \text{no} \rangle.\text{Counter}_{i+1}\langle \text{up}, \text{down} \rangle \qquad 1 \leq i < max$$
$$+\ \text{down}.\text{Counter}_{i-1}\langle \text{up}, \text{down} \rangle$$

$$\text{Counter}_{\text{max}}(\text{up}, \text{down}) \stackrel{\text{def}}{\equiv} \overline{\text{up}}\langle \text{yes} \rangle.\text{Counter}_{\text{max}}\langle \text{up}, \text{down} \rangle \qquad \text{max} \in \mathbb{N}$$
$$+\text{down}.\text{Counter}_{\text{max}-1}\langle \text{up}, \text{down} \rangle$$

4.2 Recommender System Model

This part extends the baseline model in order to include requests that originate from recommender or reviews websites. The extension takes into account the third type of request 'recommend'. This is to show that the proposed PAC scheme caters for different types of requests and different types of originating sources. Thus requests

originating from recommender websites may get higher priority as the clients would more likely to make purchases from the provider websites.

The main components that are extended from the baseline model include the specification for server, client and handler.

Server:
Unlike the base model a third counter component is added to count 'recommend' requests.

$$\text{Server} \overset{\text{def}}{\equiv} \text{new bind}, \overrightarrow{\text{up}}, \overrightarrow{\text{down}}$$

$$\text{Gatekeeper}|\text{Processor}|\text{Counter}_0\langle \text{up}_1, \text{down}_1\rangle|\text{Counter}_0\ \langle \text{up}_2, \text{down}_2\rangle|\text{Counter}_0\ \langle \text{up}_3, \text{down}_3\rangle$$

Client:
Each client receives a link from the Server which is then used to connect it with a dedicated Handler. Each client then submits various requests to the system via Handler. The Client is specified as follows:

$$\text{Client} \overset{\text{def}}{\equiv} \text{connect(link)}.\text{Client}`_0\ \langle \text{link}\rangle$$

$$\text{Client}`_i(\text{link}) \overset{\text{def}}{\equiv} \overline{\text{link}}\ \langle \text{buy}\rangle.\text{link(resp)}.\text{Client}`_{i+1}\ \langle \text{link}\rangle \qquad 0 \le i < n$$

$$+\overline{\text{link}}\langle \text{recommend}\rangle.\text{link(resp)}.\text{Client}`_{i+1}\ \langle \text{link}\rangle$$

$$+\overline{\text{link}}\langle \text{browse}\rangle.\text{link(resp)}.\text{Client}`_{i+1}\ \langle \text{link}\rangle$$

$$\text{Client}`_n\ (\text{link}) \overset{\text{def}}{\equiv} \overline{\text{link}}\langle \text{end}\rangle.0 \qquad n \in N$$

The Handler:
Each handler receives a request from its client. If the system (processor) has capacity, the request is processed, the counter is decremented, and the results of the query are passed to the client. The handler then waits for the next request to be processed.

$$\text{Handler(link)} \overset{\text{def}}{\equiv} \text{link(req)}$$

$$(\ \text{if req} = \text{end then } 0$$

$$+$$

$$\text{if req} = \text{buy then Handler}` \langle \text{link}, \text{req}, \text{up}_1, \text{down}_1\rangle$$

$$+$$

$$\text{if req} = \text{recommend then Handler}` \langle \text{link}, \text{req}, \text{up}_2, \text{down}_2\rangle$$

$$+$$

$$\text{if req} = \text{browse then Handler}` \langle \text{link}, \text{req}, \text{up}_3, \text{down}_3\rangle$$

$$)$$

Note that the specifications for the Counter and Processor remain the same as that of baseline model.

4.3 Commercial Web Portal model

This part extends the baseline model in order to include requests that originate from commercial web portals (e.g., travelsupermarket.com) and are submitted to provider web servers (e.g., http://www.britishairways.com). In the proposed PAC scheme, requests originating from such portals will get higher priority as the clients would more likely to make purchases from the provider websites.

In this model we formally specify two different servers: the provider (or seller) web server and the portal server. Each of these servers comprises same components as that of baseline model. These include: *gatekeeper, scheduler, handler, processor and counter*. We modify the formal specifications of server, client and handler. The remaining parts remain the same as that of baseline model.

Under this model the overall system, represented as eCommerce, can be formally specified as:

$$eCommerce \triangleq \text{new } P\_connect, S\_connect$$

$$ClientD|\ldots\ldots|ClientD|Server|Portal|ClientP|\ldots\ldots|ClientP$$

a) Provider Web Server:

The Provider Web Server is made up of a number of components, including: Gatekeeper, Handler, Processor and a Counter. It is specified as follow.

$$Server \triangleq new\ \overrightarrow{up}, \overrightarrow{down}, do\ Gatekeeper\_S|Processor|Counter_0\langle up_1, down_1\rangle$$

$$|Counter_0\langle up_2, down_2\rangle|Counter_0\langle up_3, down_3\rangle$$

Direct Clients:

These are the clients that directly submit requests to the provider web server. Each of these clients receives a link from the web server which is used to connect it to a dedicated handler. After the client has sent a number of fixed browse, or buy, requests, it informs the Handler_S that it has finished its requests. Handler_S represents Handler on the provider web server side. Direct Client, represented as ClientD, is formally specified as:

$$ClientD \triangleq S\_connect(link).\ ClientD`_0\langle link\rangle$$

$$ClientD`_i(link) \triangleq \overline{link}\langle Direct\_buy\rangle.link(resp).ClientD`_{i+1}\langle link\rangle \quad 0 \leq i < n$$

$$+\overline{link}\langle Direct\_browse\rangle.link(resp).ClientD`_{i+1}\langle link\rangle$$

$$ClientD`_n(link) \triangleq \overline{link}\langle end\rangle.0 \qquad n \in N$$

Gatekeeper_S:

It represents gatekeeper on the provider web server side. It deals with admission control and handles the initial direct client requests. When a client connects to the Gatekeeper_S, a fresh action is passed to both the client and a new instance of the

Handler_S, allowing these components to interact privately. Gatekeeper_S is formally specified as follows:

$$Gatekeeper\_S \stackrel{def}{\equiv} link\,\overline{S\_connect}\langle link\rangle.Gatekeeper\_S\,|Handler\_S\,\langle link\rangle$$

Handler_S:

It represents handler on the provider web server side. An instance of this component is created for each direct client that connects to the system. It starts by receiving requests from a (direct) client and terminates when the client has finished submitting requests. It passes links to the appropriate counter (specified below) which checks to see whether there is any space to accommodate the client's request. If there is space, the request is processed, the counter is decremented, and the results are passed to the client. If there is no space, the request is rejected and the client is informed accordingly. The handler then waits to process the next request. Handler_S is formally specified as follows.

$$Handler\_S(link) \stackrel{def}{\equiv} link(req)$$

$$\begin{aligned}
&(\quad if\ req = end\ then\ 0 \\
&\quad + \\
&\quad if\ req = Portal_buy\ then\ Handler_S`\langle link, req, up_1, down_1\rangle \\
&\quad + \\
&\quad if\ req = Direct_buy\ then\ Handler_S`\langle link, req, up_2, down_2\rangle \\
&\quad + \\
&\quad if\ req = Direct_browse\ \ then\ Handler_S`\langle link, req, up_3, down_3\rangle \\
&)
\end{aligned}$$

$$Handler\_S`(link, req, full, dec, connect) \stackrel{def}{\equiv} full(ans)$$

$$\begin{aligned}
&(\\
&if\ ans = yes\ then\ \overline{link}\langle reject\rangle.Handler_S\langle link\rangle \\
&+ \\
&if\ ans = no\ then\ Process\langle link, req, dec\rangle \\
&)
\end{aligned}$$

$$Process(link, req, dec) \stackrel{def}{\equiv} \overline{connect}\langle req\rangle.connect(resp).\overline{dec}.\overline{link}\langle resp\rangle.$$
$$Handler\_S\langle link\rangle$$

In addition, the provider server has the same processor and counter definitions as the baseline model (4.1).

b) Portal Web Server

The Portal Web Server is made up of a number of components, including: Gatekeeper, Handler, Processor and a Counter. It is specified as follow.

$$Portal \stackrel{def}{\equiv} new\ up_1, down_1, do\ Gatekeeper\_P|Processor|Counter\langle up_1, down_1\rangle$$

Portal Clients:

The portal client receives two links, one from the portal server which connects it with a dedicated Handler_P to process the browse requests, and another link from the web server which connects it with a dedicated Handler_S to process the buy requests. When the portal client finishes sending a number of browse and buy requests, it informs the specified handler that it has finished its requests.

Portal Client, represent as *ClientP*, is formally specified as follows.

$$ClientP \triangleq P\_connect(link).S\_connect(link1).ClientP_0`\langle link, link1\rangle$$

$$ClientP`_i(link, link1) \triangleq \overline{link1}\langle Portal\_buy\rangle.link1(resp).ClientP`_{i+1}\langle link, link1\rangle$$
$$+ \overline{link}\langle browse\rangle.link(resp).ClientP`_{i+1}\langle link, link1\rangle \quad 0 \leq i < n$$

$$ClientP_n`(link, link1) \triangleq \overline{link}\langle end\rangle.\overline{link1}\langle end\rangle.0 \quad n \in N$$

Gatekeeper_P:

It represents gatekeeper on the portal web server side. This is formally specified as follows.

$$Gatekeeper\_P \triangleq link\ \overline{P\_connect}\ \langle link\rangle.Gatekeeper\_P\ |Handler\_P\ \langle link\rangle$$

Handler_P:

This represents handler on the portal web server side. It is specified in a similar way to that of Handler_S, detailed above.

$$Handler\_P(link) \triangleq link(req)$$
$$($$
$$if\ req = end\ then\ 0$$
$$+$$
$$if\ req = browse\ then\ Handler\_P`\langle link, req, up_1, down_1\rangle$$
$$)$$

$$Handler\_P`(link, req, full, dec, connect) \triangleq full(ans)$$

$$($$
$$if\ ans = yes\ then\ \overline{link}\langle reject\rangle.Handler\_P\langle link\rangle$$
$$+$$
$$if\ ans = no\ then\ Process\langle link, req, dec\rangle$$
$$)$$

$$Process(link, req, dec) \triangleq \overline{connect}\langle req\rangle.connect(resp).\overline{dec}.\overline{link}\langle resp\rangle.$$
$$Handler\_P\langle link\rangle$$

In addition, the portal server has the same processor and counter definitions as the baseline model (4.1).

5. Implementation of the Prototype System

The formally specified models of the proposed PAC scheme are implemented in Java. It follows the techniques in [8] with those aspects of the behaviour that were abstracted from in the model being fully defined such as changing priorities and adding client histories. It also provides support for large number of clients. Java was chosen as the implementation language as it is easy to extend the proposed simulation (of the PAC scheme) by adding new components. In addition, Java language is able to have concurrent systems that closely follow π-calculus specifications which are presented in section 4 above. Furthermore, Java facilitates communication between server and client using Transmission control protocol (TCP) which is the underlying protocol for most of the web applications.

The multi-actor simulation was developed to reflect the target models as accurately as possible for all experiments. The proposed models were solved using multi-actor simulation. The structure of the simulation is based on the state of the proposed models which mimic commercial websites setup that include servers and clients. The prototype system randomly generates a number of requests of a particular type. The system is implemented using JCreator LE 4.00 on Microsoft Windows XP. TCP connection is used between the components of the system in order to ensure reliable communication and to avoid loss of messages. The prototype system also takes into account the mutual exclusion of the shared resources such as processor or memory required by the different types of requests. The prototype system guarantees the exclusive access to shared resources so that requests do not interfere with each other.

The following section presents various experiments which are carried out using the Java-based prototype system.

6. Evaluation of the Proposed Scheme

This section evaluates the performance of the proposed PAC scheme through a series of experiments.
Various simulation experiments are conducted using the prototype system. The experiments cover a wide range of input parameters, such as number of clients, the types of requests, server capacity, originating sources of requests, etc. The experiments take into account all the important parameters of performance metrics which would have the maximum impact on the performance in the design and analysis of a commercial web server. These include, the *average response time* of requests and the *number of dropped requests* whenever they exceed the capacity of a web server. That is, if the number of arriving requests is more than they can be accommodated in the buffers (or queues as described above), then the Gatekeeper will not allow them to enter into the system. Instead, they will be dropped or rejected. The rejection policy is based on the priority of requests. High priority requests will have lower rejection rate than lower priority requests. We measure these two (response time and dropped requests) as these have significant effects on the web server performance as well as the clients. If clients' requests (of low priority) are frequently rejected then they will not be using the websites. Similarly, if the response time is very low then it may push clients to use the websites. Thus the proposed scheme keeps the balance between improved response time and reduced dropped requests.

The results also give valuable insight into the performance studies of real commercial web servers. For example it can be used to run real-time experimentation tests to improve websites performance. These tests involve testing a new treatment or

feature (added to website) for a limited time of a few days or a week. The system will measure a range of parameters such as revenue, session time, session length, etc.

6.1 Performance Evaluation of Recommender System

Experiments are conducted using the prototype system described in the previous section. The prototype implements various buffers in order to accommodate incoming requests of different types (as described in previous section). Each of the buffer capacity is set such that it can hold a maximum of 100 requests - i.e., the maximum value of the Counter goes to 100. However this can be easily increased. First, we conducted several experiments with multiple adjustments to set the best size for the buffer. One of the performance metrics is chosen to see the effect of different buffer sizes on obtained results. We found that the best buffer size is 100 because it gives an acceptable arrival rate and lower rejection of requests. It also gives better results as there are enough places to accept incoming requests and to balance the obtained results during the simulation. The service time for each type of request is chosen randomly (through exponential distribution) with a mean of 10ms. To study the effect of changing the load on the performance metrics, the system examined when the service rate is similar for all types of requests with burst traffic which increased confidence in the proposed simulation. This is due to its flexibility and support for the use of several types of traffic models. In other words, it is easy to adopt a new traffic model, by simply changing the instantiation routine of the client object while keeping intact the other parts of simulator.

For the recommender system experimentations, three types of requests are considered: browse, recommend and buy. Using the prototype these requests are randomly generated from different clients. The number of different types of request varies; for example, 20% buy requests, 30% recommend requests and 50% browse requests. Generally, there are more 'browse' requests than 'buy' or 'recommend' requests. The proposed scheme gives higher priority 'buy' and 'recommend' requests compared to 'browse' as the former are more important to the service provider.

Figure 3 shows the results when the three different types of requests arrive at the system. In this case, low priority requests (e.g., browse) are not totally dropped. Instead they are allowed to enter the system. However, priority is given to buy and recommend' requests. The results show a clear improvement in the performance of high-priority requests over medium- and low-priority ones. In this case, the system gives a small delay to processing of low priority (browse) requests rather than rejecting them, thus it keeps low priority customers on the system. However, this additional delay has a large impact on the client response time. That is, it may increase the response time of the high priority requests.
It appears that the response times are low for small numbers of clients. The response time is related to the length of the buffer. Note that the sharp increase in average response time occurs when the server is fully utilized and the buffer starts to fill particularly in the beginning of the simulation.

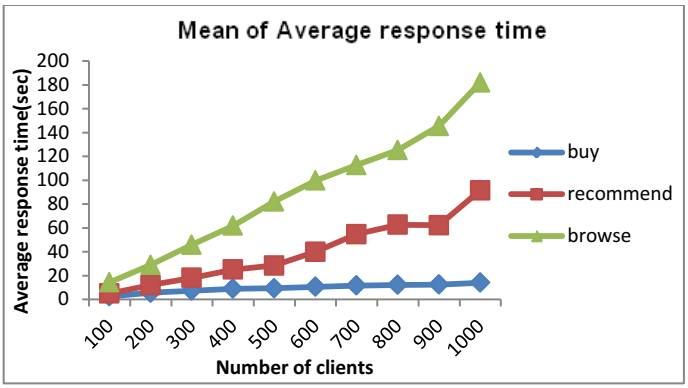

Fig. 3. Average response time in Low priority fair Model

Taking into account the average response time (Figure 3), the next experiment in Figure 4 shows the percentage of dropped requests when system does not have capacity to accommodate all the different types of requests. Though compared to the buy requests, browse and recommend' requests have higher drop rates but this is still better than Figure 6, which significantly drop low priority requests. These results (Figures 3 and 4) show that there should be a tradeoff between maintaining appropriate response time and dropped rate. It is not recommended to improve response time at the expense of increased dropped rate as this will push clients to move away from the website if their (low priority) requests are dropped continuously.

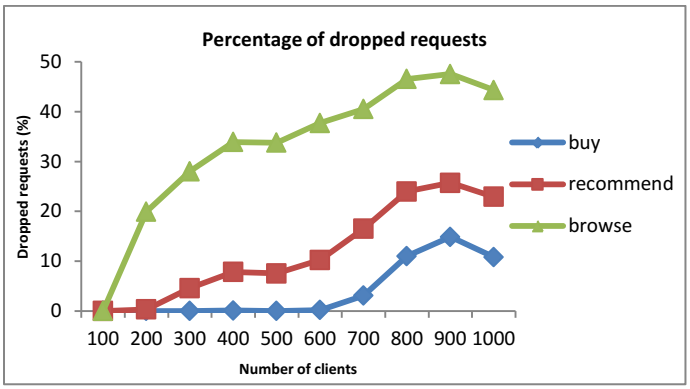

Fig. 4. Dropped requests in Low priority fair Model

Figure 5 shows the response time of requests in which a large number of low priority requests is rejected. The increase in the response time of buy request is due to the fact that most of the requests entering the system are of high priority (buy) requests. However, due to the large percentage of low and medium-priority requests being dropped, these will have a low average response time. With more generated requests, the number of low and medium-priority requests completed will decrease because most of these requests will be rejected. The observed response time (in Figure 5) as seen by the client side can give clear information about the server performance. When the server is close to 100% utilized, it shows that the system gives best value by completing most of the highest-priority buy requests.

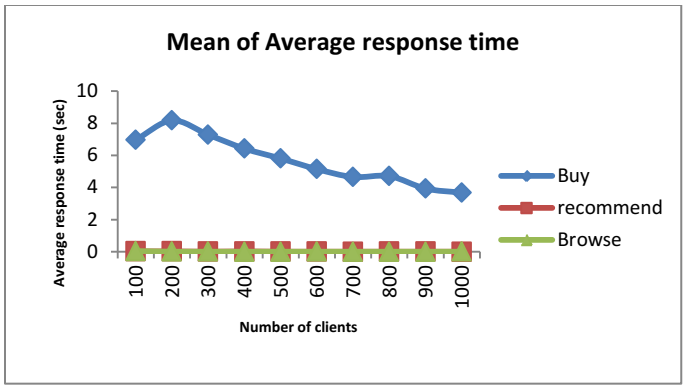

Fig. 5. Average response time in Review Model

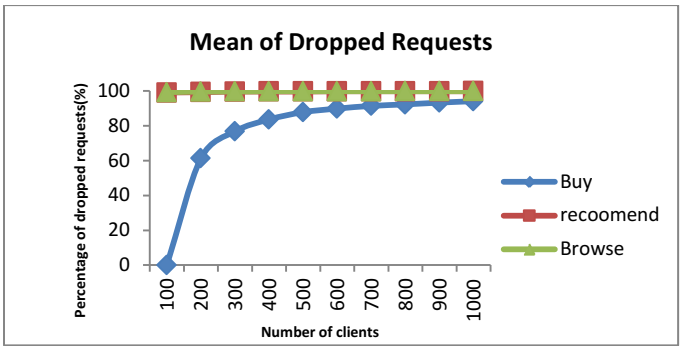

Fig. 6. Dropped requests in Review Model

Figure 6 presents the effect of an increasing number of clients on the percentage of dropped requests for each type. It can be seen that the occurrence of the request being dropped steadily increases for all types upon an increase in the number of clients. This is clearly due to the finite capacity of the buffer for incoming requests.
There are rejected buy requests under heavy load conditions at the same point above 100 clients, so that the average response time becomes small (see Figure 5). However, there is a huge number of other rejected low-priority requests. The 'browse' requests start to be rejected a little earlier than 'recommend' ones because the processor is being occupied with 'recommend' requests and occupied fully with 'buy' requests which have higher priority.

6.2 Performance Evaluation of Web Portal Model

In these experiments client requests are configured using two different TCP ports; one port connects clients with the provider web server and the other connects with the portal web server. Two sets of experiments are designed using different workload: In the first set of experiments baseline model is used. It gives highest priority to the client requests which come from portal web sites and to complete their purchase (buy requests) at provider web server. These requests are called "Portal_buy". The second level priority is given to "Direct_buy" who comes directly to the web sites to purchase, and the lowest priority is given to "Direct_browse" who browse directly at provider website.
In the second set of experiments the fair low priority model is used which gives a small delay to processing "Direct_browse" requests rather than rejecting them. Note that the portal web server serves only browse requests sent by portal clients, and these are called "Portal_browse".

Set 1: Experiments

Results of these experiments are shown in Figures 7 and 8. Figure 7 shows the average response time of all types of requests that serve at the provider web server and portal web server. As expected, response times are low for small numbers of clients, whereas large numbers of clients (direct and portal) produce a heavy load on the web server, resulting in a slightly increased average response time for high-priority requests because the web server processes most Portal_buy requests. However, due to the large percentage of low-priority requests being dropped, these will have a low average response time. Average response times (Figure 7) are compatible with the dropped requests (Figure 8) at the same point of 300 clients the dropped requests starts to increase dramatically thus causing lower response time.

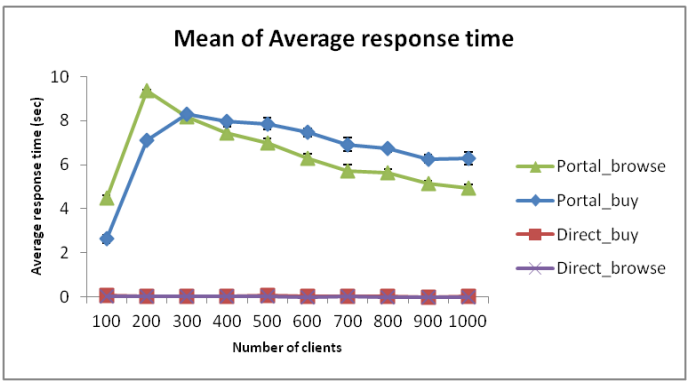

Fig. 7. Average response time in baseline Portal Model

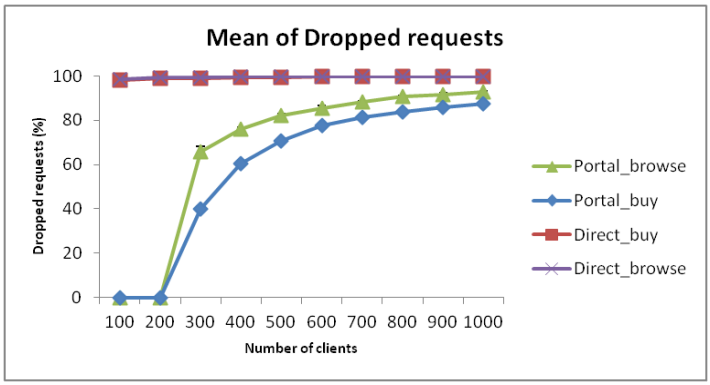

Fig.8. Mean of dropped requests in baseline Portal Model

Moreover, Figure 7 shows that a higher number of clients produces a higher average response time of Portal_Browse because the portal server receives a large number of browse requests and there is not any competition to process Portal_Browse requests. As shown in Figure 8, with more generated requests the number of Portal_Browse requests completed will decrease because most of these requests will be rejected. This diminishes response time because there is no time spent on rejected requests.
As shown in Figure 7 most handlers process Portal_buy requests first, then Direct_buy and then Direct_browse. With more requests, or with a fully-utilized server, the percentages of all types of completed requests will decrease because most

of these requests will be rejected by the web server. The results indicate that the highest-priority customers take less time to process because they are prioritized, while the lowest-priority customers lose more requests.

Set 2: Experiments

Due to the high percentage of drop in low priority requests (Direct_buy and Direct_browse), the fair low priority model is used in order to reduce the bias towards such requests. Figure 9 shows the average response time of all types of requests that are serviced at the provider web server and portal webserver. As expected, response times are low for small numbers of clients, whereas larger numbers of clients (direct and portal) produce a heavy load on the web server, resulting in a slightly increased average response time for high-priority requests. This is because the web server processes most "Portal_ buy" requests. Average response times (Figure 9) are compatible with the dropped requests (Figure 10) at the same point of 300 clients the dropped "Portal_browse" requests starts to increase dramatically. This causes lower response time because there is no time spent on rejected requests.

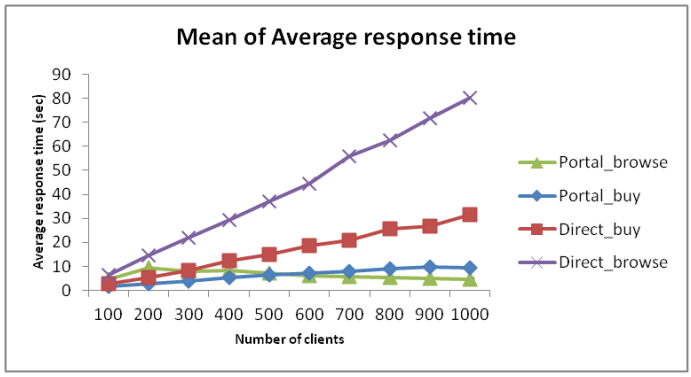

Fig.9. Mean of dropped requests in fair low priority Portal Model

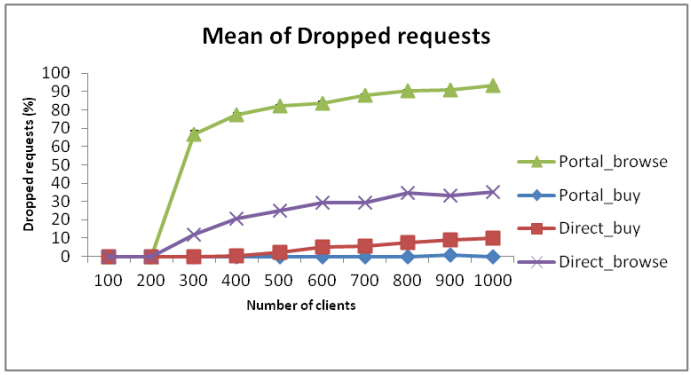

Fig.10. Mean of dropped requests in fair low priority model Portal Model

As shown in Figure 10 most handlers process Portal_buy requests first, then Direct_buy and then Direct_browse. With more requests, or with a fully-utilized server, the percentages of all types of completed requests will decrease because most of these requests will be rejected by the web server. With more generated

"Portal_browse", the portal server will reject most of them according to the status of buffer (being fully occupied). Note that the Portal_browse results are not effective with the different basic models because it considers only the web server.

It is observed from above results that lower drop rates of all requests occur because the fair low priority model avoids rejecting the low-priority requests and also gives the system more time to process the higher-priority customers.

6.3 Summary of experimental results

The performance has been evaluated by taking account of different situations and conditions, for example, varying the number of clients, changing the traffic flow or varying the buffer size and changing the models. The results obtained clearly demonstrate how different load settings can provide different response times and the rate of dropped requests. Overall, the results demonstrate the effectiveness of the proposed models and show a marked improvement of the performance of high-priority requests. Though low-priority requests experience reduced performance as well as rejected requests, there is a clear improvement in the performance of low-priority requests using the Low Priority Fair Model. It reduces the rejected requests without any effect on the performance of high-priority requests. It is also observed that the response time is consistent with dropped requests.

7. Conclusions

In this paper, we developed PAC scheme for commercial web servers. In commercial websites, some requests, e.g. payments, are generally considered more important than others, i.e. search or browse, due to their revenue-raising capabilities. The aim of PAC scheme is to increase the performance of high-priority requests but at the same time reduce their impact on the low priority requests. The proposed PAC scheme combines several techniques, including: admission control mechanism, session-based admission control, service differentiation, request scheduling, and queuing model-based approaches. It is formally designed using widely used formally specification language of π-calculus. The proposed scheme is incrementally designed starting with the baseline model and extending it to more complex models of recommender systems and portal websites. Based on formal specification the PAC scheme is implemented as a prototype system using Java. We analyzed and evaluated the PAC scheme through various experiments. The experiments covered a wide range of input parameterizations and gave insight into the different aspects of the performance of web servers. The experiments showed the PAC scheme improved the performance of high priority requests without causing severe impact on low priority requests.

References

[1] Menascé, D,A., Almeida, V,A, F., Fonseca, R., and Mendes, M.A. (1999) A Methodology for Workload Characterization of E-commerce Sites. ACM Conference on Electronic commerce, Denver, Colorado, USA, pp. 119-128.

[2] Jakob Nielsen "Why People Shop on the Web" http://www.useit.com/alertbox/990207.html

[3] IBM, developer Works, Available from: http://www.ibm.com/developerworks/websphere/newto/.[Accessed 30th April 2013]

[4] Ghandeharizadeh.S,Yab.J andBarahmand.S.(June 2012) COSAR-CQN: An Application Transparent Approach to Cache Consistency, the Twenty First

International Conference On Software Engineering and Data Engineering,LosAngeles,California,June27-29,2012.
[5] Jilly,K.,Juiz,C., Thomas,N., and Puigjaner,R.(Septemper,2012) Adaptive admission control algorithm in a QoS-aware Web system ,Information sciences, volume 199,pages 58-77.
[6] Z Zhang, HP Wang, JT Liu and YF Hu.(2012) Performance Analysis and Optimization of Network Server Based on Node Cache, Computer Engineering, *1*, 003., Computers and communications,Proceedings. ISCC 2000. Fifth IEEE Symposium, pp.359 - 363.
[7] Chollette, Chude C., et al. "A queue scheduling approach to quality of service support in Diff-Serv networks using fuzzy logic." *Computer and Communication Engineering (ICCCE), 2012 International Conference on*. IEEE, 2012.
[8] Milner,R. (1999) Communicating and Mobile systems: the π-Calculus. Cambridge University Press.
[9] Crovella, M., Frangioso, R., Harchol-Balter, M.(11-14 October 1999) Connection scheduling in web servers, ACM, Second symposium of Internet Technologies and System (USITS'99),Boulder, CO,USA, volume 2.
[10] Rawat, M., Kshemkayani, A. (16-18 April 2003) SWIFT: Scheduling in web servers for fast response time, Second IEEE International Symposium on Network Computing and Applications (NCA 2003),Cambridge, MA, USA,pp.51-58.
[11] Chuan, Yue., Haining, Wang.(2009) Profit-aware overload protection in E-commerce Web sites, Journal of Network and Computer Applications, 32, pp. 347– 356.
[12] Bertini, L., Leite, J., Mossé, D.,(2010) Power and performance control of soft real-time web server clusters, Information Processing Letters,2010, 110: 767–773.
[13] Almeida, J., Almeida, V., Ardagna, D.,Cunha, I., Francalanci, C., and Trubian, M.,(2010) Joint admission control and resource allocation in virtualized servers, Journal of Parallel and Distributed Computing,2010, 70, 344_362.
[14] Younas,M., Awan,I.,Chao, K.-M., Chung, J.-Y.(2008) Priority scheduling service for e-commerce web servers, Journal of Information Systems and E-Business Management 6(1),69-82,2008.
[15] Urgaonkar, B., and Shenoy, P.(2008) Cataclysm: Scalable overload policing for internet applications, Journal of Network and Computer Applications (JNCA), 2008, 31:891-920.
[16] Bhatti, N., and Friedrich, R., (1999) Web server support for tiered services, IEEE Network, 1999, 13(5):64-71.
[17] Verma, A., and Ghosal, S.(2003) On admission control for profit maximization of networked service providers, 12th International World Wide Web Conference (WWW'03), Budapest, Hungary, 2003, p 128-137.
[18] Chen, H., and Mohapatra, P.(2003) Overload control in QoS-aware web servers, Computer Networks, 2003,42(1):119-133.
[19] Alonso, J. ,Guitart, J., and Torres, J. (2007) Differentiated Quality of service for e-Commerce Applications through Connection Scheduling based on System-Level Thread Priorities, Parallel, Distributed and Network-Based Processing 15th EUROMICRO International Conference, 7-9 Feb. 2007, pp.72 – 76.
[20] Muppala, S., Xiaobo, Zhou. (2011) Coordinated session-based admission control with statistical learning for multi-tier internet applications, Journal of Network and Computer Applications, volume 34, pp.20–29.
[21] Abdelzaher, T.,Shin, K., and Bhatti, N.(2002) Performance guarantees for web server end-systems: A control-theoretical approach, IEEE Transactions on Parallel and distributed Systems,2002, 13(1):80-96.

[22] Almeida, J., Almeida, V., Ardagna, D., Francalanci, C., and Trubian, M. (2006) Resource management in the autonomic service-oriented architecture. Third International Conference on Autonomic Computing (ICAC'06), Dublin, Ireland, 2006, pp,4–92.
[23] Yang, Changwoo, et al. "Many flows asymptotics for SMART scheduling policies." *Automatic Control, IEEE Transactions on* 57.2 (2012): 376-391.
[24] Brafton Editorial ,"89 percent of consumers use search engines for purchase decisions", http://www.brafton.com/news/89-percent-of-consumers-use-search-engines-for-purchase-decisions, Published on February 2, 2012.
[25] Shari Thurow,"Search Engine Marketing For Travel-Related Sites", http://searchenginewatch.com/article/2065924/Search-Engine-Marketing-For-Travel-Related-Sites,October 25, 2004.
[26] Jessica Davies, " 70% of people abandon online baskets prior to purchase", http://www.thedrum.com/news/2013/07/02/70-people-abandon-online-baskets-prior-purchase#IiPv3iDdkohZTJwP.99, 2 July 2013.