A class-based scheme for E-commerce web servers: Formal specification and performance evaluation

D.R.W. Holton\(^a\), I. Nafea\(^a\), M. Younas\(^b\), I. Awan\(^a,\)\(^*\)

\(^a\) Department of Computing, The University of Bradford, Bradford BD7 1DP, UK
\(^b\) Department of Computing, Oxford Brookes University, Oxford OX33 1HX, UK

**Abstract**

This paper is an investigation into the performance of E-commerce applications. E-commerce has become one of the most popular applications of the web as a large population of web users is now benefiting from various on-line services including product searches, product purchases and product comparison. E-commerce provides users with 24-7 shopping facilities. However, the consequence of these benefits and facilities is the excessive load on E-commerce web servers and the performance degradation of E-commerce (eCom) requests they process. This paper addresses this issue and proposes a class-based priority scheme which classifies eCom requests into high and low priority requests. In E-commerce, some requests (e.g. payment) are generally considered more important than others (e.g. search or browse). We believe that by assigning class-based priorities at multiple service levels, E-commerce web servers can perform better and can improve the performance of high priority eCom requests. In this paper, we formally specify and implement the proposed scheme and evaluate its performance using multiple servers. Experimental results demonstrate that the proposed scheme significantly improves the performance of high priority eCom requests.

© 2008 Published by Elsevier Ltd.

1. Introduction

E-commerce is becoming increasingly popular in the global economy as more and more business transactions are conducted over the web. The boom in E-commerce is driven by the ubiquity of the web, the 24-7 on-line services, and the lower costs—customers can often make on-line purchases at a lower cost and at a time suited to them, provided they are connected to the Internet.

One of the crucial issues in the E-commerce is to provide an appropriate level of quality of service. E-commerce service providers strive to provide quality-based services by investing significant amount of money and resources in the E-commerce technologies such as web design, user interfaces, advertisement, security and reliability, and web server performance. In this paper our research focuses on the performance of E-commerce web servers. Performance plays a key role in the provision of quality-based services. Poor performance has negative effects on the image of the business. For example, if a web site takes 1 min to load, it is quite possible that user will abandon that site for an alternative, faster, one.

\(^*\) Corresponding author.

_E-mail addresses:_ d.r.w.holton@comp.brad.ac.uk (D.R.W. Holton), i.nafea@brad.ac.uk (I. Nafea), m.younas@brookes.ac.uk (M. Younas), i.u.awan@brad.ac.uk (I. Awan).
payment requests should be prioritised over browsing requests. We believe that by assigning class-based priorities at multiple service levels, E-commerce web servers can perform better and can improve the performance of high priority eCom requests without causing adverse effects on low priority requests.

The scheme proposed above is based on our previous work (Younas et al., 2008) and extends it as follows:

1. **Formally specifying the proposed scheme**: The specification process allows us to rapidly investigate a number of different protocols, as well as providing an architectural prototype for the implementation (Rodrigues and Barbosa, 2005). In addition, the architecture of the implementation can be generated automatically from the specification (Gilmore et al., 1996).

2. **Development of a prototype system**: Our previous work was based on analytical modelling and simulation results. This paper develops a prototype system which reflects the requirements of real world eCom applications.

3. **Multiple queues**: performance of the proposed system is evaluated using multiple priority queues. Our previous work was based on a single queue system. We conduct a number of experiments to rigorously evaluate the performance of the proposed scheme. Experimental results show that the proposed scheme significantly improves the performance of high priority eCom requests and also reduces the number of high priority requests that are dropped.

The remainder of the paper is structured as follows. Section 2 presents a generalised architecture of E-commerce applications. Section 3 describes related work. Section 4 presents the formal specification of the scheme and describes its implementation. Section 5 analyses the results obtained from the implementation and finally Section 6 presents our conclusions.

2. **Architecture of E-commerce applications**

This section presents a generalised architecture of E-commerce applications. The aim is to facilitate the understanding of the working mechanism of E-commerce applications.

In E-commerce, users interact with E-commerce web servers through a series of requests in order to acquire required information or buy products. For example, to find a mobile phone from a web site, a user first either enters a keyword or clicks on a category link of a required mobile phone. The user's request is sent to the web server which in turn passes the request to the application server and then to the database server. As shown in Fig. 1, E-commerce applications are generally implemented by a multi-tier architecture that comprises client systems, network, web servers, application servers and data servers. Web servers typically serve static content such as HTML pages. Application servers (e.g., BEA WebLogic, IBM WebSphere) are commonly used to generate dynamic web contents by running scripts written in a number of languages such as active server pages (ASP), java server pages (JSP), and Perl. Scripts execute the necessary logic to process customers’ requests by contacting various resources in order to retrieve, process, and format the requested content into customer deliverable web pages. Our work is not concerned with the performance aspects of the application server or the database server. It focuses on web server performance and how it can be improved by implementing a class-based priority scheduling scheme.

3. **Related work**

This section reviews related work on the scheduling mechanisms that have been developed in order to improve the performance of web servers. Though there exist other mechanisms (for example Menasce, 2002; der Meer et al., 2004) for improving web server performance, our review is targeted at scheduling mechanisms as these are closely related to the work presented in this paper.

Elnikety et al. (2004) aim to improve the performance of E-commerce applications. This work proposes a method for admission control and request scheduling for multi-tiered E-commerce applications. This method differentiates between the different types of requests and devises a preferential scheduling policy in order to assign different priorities to different requests. A proxy server, called Gatekeeper, is developed that enables admission control and implements the preferential scheduling policy. The preferential scheduling is based on the shortest job first (SJF) policy. However, such a policy may fail to improve the response time if E-commerce requests are homogeneous, that is, requiring the same service time.

Harchol-Balter et al. (2003) employ a preemptive version of SJF scheduling, called shortest remaining processing time (SRPT) first priority. SRPT is used to improve the performance of web servers. However, this work considers static web pages such that priority is given to requests for small files or requests with shortest remaining file size.

McWherter et al. (2004) propose a priority mechanism for transactions in classical database systems. This work presents a detailed analysis of the resource utilisation by transactions in a database system. It also improves the performance of high priority transactions in classical database system.

Kamra et al. (2004) propose a control-theoretic approach for multi-tiered web applications. It aims to prevent overload and to ensure high throughput and to maintain absolute response time. The proposed approach is implemented as a proxy system called Yaksha—which is claimed to be non-invasive and which avoids frequent operator intervention.

Menasce et al. (1999) propose a customer behaviour model graph (CBMG) that describes the behaviour of customers who follow similar navigational patterns in submitting requests to E-commerce web sites. In these sites on-line shoppers issue requests such as browse, search, and pay. CBMG is used to describe the sequence of such requests. CBMGs are constructed by analysing logs of an E-commerce site that contains information related to a user’s profile based on their previous navigation patterns. Different users are characterised by different CBMGs. For example, one CBMG can be constructed for occasional buyers who usually use E-commerce site to find information such as air fares, itineraries, and books prices, without buying anything. Another...
CRMG can be constructed for users who have high probability of buying products from the E-commerce site. CBMGs are useful to determine the behaviour of customers visiting an E-commerce site. For example, when a customer starts navigating a web site, the web server can use the profile information (stored in the log file) and assigns different priorities based on a user profile. However, it incurs processing overhead in constructing the CBMG using the past information stored in the log files that describes the customers' profiles—a CBMG is constructed even if a customer visiting a web site does not buy items. Another alternative is to use registration information to classify customers into occasional buyers or heavy buyers. That is, registered users are more likely to buy (heavy buyers) as compared to non-registered users who are less likely to buy (occasional buyers). However, it is unrealistic to assume that registered users will buy items each time they visit an E-commerce web site.

Zhou et al. (2004) propose a two-dimensional (2D) service for classifying on-line transactions into inter-session and intra-session transactions. The former provide differentiated QoS to the sessions according to customer classes, while the latter provide differentiated QoS according to states of a particular session.

Singhmar et al. (2004) propose an LIFO-Pri priority scheduling scheme in order to give service priority to revenue generating (such as payment) requests over the browsing requests. This scheme is based on a large number of queues which are extremely difficult to manage. The proposed scheme works by moving revenue generating requests from one queue to another queue based on its current state during its processing. This requires that requests are tracked throughout their entire execution. It may be manageable for small numbers of requests but will show performance degradation for larger numbers. Our previous work Aw and Younas (2004), Younas and Aw (2003) employs active network priority scheduling mechanisms in order to improve the performance of transaction commit protocols in Web-database applications. These approaches gives preferential treatment to the processing of decision messages (such as transaction commit, abort, compensate) over data related messages.

4. The proposed scheme

The main attribute lacking in the existing approaches is simplicity. This paper investigates a class-based priority scheme for multiple classes of web requests with the intention of providing a simple, easily implemented method for reducing the response time for high priority request. The scheme works as follows (Fig. 2):

1. Requests are classified into two types, namely search or browse requests and payment requests, and directed to the buffer devoted to holding that type of request (thus there are two distinct buffers).
2. If the buffer containing high priority requests is not empty, the first request in that buffer is processed.
3. If the buffer containing high priority requests is empty, then process the first request from the low priority buffer.

4.1. Formally specifying the scheme

The π-calculus (Milner, 1999) was used to specify the scheme. The motivation for constructing a formal specification was to allow us to rapidly investigate a number of alternative schemes, and to act as a framework for the implementation. The π-calculus was chosen because of its support for mobility (to be addressed in future work).

4.1.1. The client

In order to collect experimental results of the performance of the scheme, it was necessary to specify a client component. The behaviour of the client is independent of the protocol used to process its queries, and is simple: it sends a number of browse and revenue generating queries to a server, waiting for a response to each query before sending the next. After all n queries have been processed the client terminates.

\[ \text{Client}(query, response) \]

The client sends its response action as a parameter so it can receive a reply from the server once the query has been processed. The specification shows the client making a non-deterministic choice between browse and buy queries; this is appropriate since the purpose of this specification is to describe how the components 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.

4.1.2. Server with a single finite buffer

Before it can be shown that the proposed scheme improves on existing practice, it is necessary to provide a point of comparison. This is done by examining the performance of a simple server which serves requests in a strictly FIFO order, regardless of their priority.

The server is made up of a buffer component and a processor component. The processor receives queries from the buffer, processes them, and responds to the appropriate client:

\[ \text{Processor}(port) \text{ def } \text{port(type, who), who, Processor(port)} \]

As with the client, details of the processing of the query are abstracted from.

The buffer maintains a sequence of the queries in the order they arrive. If the buffer is full (i.e. has m elements in it) when a new query arrives, then the new query is dropped. A sequence is an ordered collection of items enclosed in square brackets. The concatenation operator (\(\cdot\)) which joins two sequences is used to enforce an FIFO ordering of queries.

The components defined above are composed to define the server, using a new action to connect them:

\[
\text{Server}(\text{request}) \equiv \text{new connection Processor}(\text{connection})
\]

\[
| \text{Buffer}_0(\text{request}, \text{connection}, [])
\]

4.1.3. Server with two finite buffers

The specification of the proposed scheme requires a new component (the Gatekeeper) whose purpose is to direct incoming queries to the correct buffer:

\[
\text{Gatekeeper}(\text{request}, \text{query}, \text{type}, \text{who}) \equiv \text{request}(\text{type}, \text{who})
\]

\[
\text{if type = browse then } \text{Buffer}_\text{browse}(\text{type}, \text{who}); \text{Gatekeeper}(\text{request}, \text{query}, \text{type}, \text{who})
\]

\[
\text{if type = buy then } \text{Buffer}_\text{buy}(\text{type}, \text{who}); \text{Gatekeeper}(\text{request}, \text{query}, \text{type}, \text{who})
\]

The buffer defined in Section 4.1.2 can be re-used, but a new definition for the processor is required. The processor receives a request from either queue, processes it, and responds to the client originating the request. The concrete details of how the priority scheme is implemented is again left to the implementation.

\[
\text{Processor}(\text{browse, buy}) \equiv \text{process} \text{type, who} \quad \text{who} \quad \text{Processor}(\text{browse, buy})
\]

\[
+ \text{buy}(\text{type, who}); \text{who} \quad \text{Processor}(\text{browse, buy})
\]

The components are composed as follows to create the new server:

\[
\text{Server}(\text{request}) = \text{new buffer browse, buffer buy, browse}
\]

\[
\text{Gatekeeper}(\text{request, buffer browse, buffer buy})
\]

\[
| \text{Buffer}_\text{browse}(\text{browse, buy}, []) | \text{Buffer}_\text{buy}(\text{buffer buy, buy}, [])
\]

\[
| \text{Processor}(\text{browse, buy})
\]

Note that the interface (i.e. the action request) to this version of the server is identical to the version with a single buffer; the client would not be able to determine which version it was interacting with, yet the improved protocol gives better utility to the operators of the server.

4.1.4. Defining the system

A population of clients and a single server are composed together, connected by a single action service used to request a service from the server, and a vector of actions reply to allow a response to be made to individual clients.

\[
\text{new service, reply}
\]

\[
\text{Client}(\text{service, reply}_1) | \text{Client}(\text{service, reply}_2) | \ldots | \text{Client}(\text{service, reply}_k) | \text{Server}(\text{service})
\]

As described above, either definition of the Server component could be used in this composition, allowing for a true comparison of the performance of the two schemes.

4.2. Implementing the scheme

An implementation of the π-calculus specification was constructed in Ada following the techniques from Gilmore et al. (1996), with those aspects of the behaviour that were abstracted from in the specification being fully defined. Ada was chosen as the implementation language for pragmatic reasons, and an implementation in Java is also being constructed.

4.2.1. The client

The client was the only contrived part of the implementation; the other components described below could be used without modification in a production version of the protocol.

The non-deterministic choice in the specification was transformed into a probabilistic choice, with 80% of queries being browse requests. It was decided that a client would terminate after generating 400 requests. Performance results were obtained with populations of up to 50 identical clients running concurrently. The client was augmented to include calls to a reporting object when its queries were processed or rejected.

4.2.2. Server with a single finite buffer

The specification of this component is conveniently translated because Ada allows nesting of tasks (active processes) within other tasks, allowing the Processor and Buffer components to be translated as tasks without them being visible to the clients. A thread-safe bounded buffer component was used to implement the sequence from the specification; given the relatively small population of clients, the size of the buffer was fixed at 16 queries.

4.2.3. Server with two finite buffers

As above, the ability to nest tasks within tasks made the implementation of this component convenient, and the buffer task from the previous section was re-used in defining the separate browse and buy buffers. The capacity of these buffers was set to eight each, ensuring the total amount of buffering was the same as above.

The Gatekeeper component was very easy to implement using Ada’s enqueue statement: depending on the type of query received the request was enqueued to the appropriate buffer.

The implementation of the Processor component was the single inelegant part of the entire process. The specification shows it making a non-deterministic choice between processing a buy or a browse request which was implemented as a priority choice: if a buy query is available then process it, otherwise process a browse query. Since the processor should never be waiting for one type of query to appear if there is another type available to process, this had to be programmed as a “busy-wait” loop. Under low load conditions this would mean a wasteful use of computing resources in just testing whether a query was available, but under more realistic conditions this would not cause problems since there would always be queries to process.

Note that this version of the server was substituted for the previous version with no modifications to the client required.

5. Experimental results

The first results (Fig. 3) compare the throughputs achieved with both schemes to ensure that the priority scheme did not introduce excessive processing overheads. As expected, the throughput reaches a maximum when the arrival rate ensures...
that the processor is fully utilised. There is no significant difference in the results obtained from both schemes, indicating that the priority scheme does not add an extra computational burden. Note that the arrival rates were varied by increasing the number of clients generating queries.

Results for the time for a query to be completed against arrival rate for the simple scheme are shown in Fig. 4(a). Since browse and purchase queries are not separated, it is no surprise that their response times are the same. As expected, the response time is related to the length of the buffer. Note the sharp increase in response time that occurs when the processor is fully utilised and the buffer starts to fill. After this point the rate of increase of response time slows, which is explained by the increasing number of requests dropped by the scheme because the buffer is full (Fig. 4(b)). Note that in Fig. 4(b), queries are not dropped until the processor is fully utilised. The difference in the slopes of the lines is explained by the difference in the probabilities with which the different types of queries are generated.

The results of response times for queries for the priority scheme (Fig. 5(a)) show an enormous improvement for purchase

![Fig. 3. Comparison of throughputs.](image)

![Fig. 4. Results from the simple scheme: (a) response times for simple scheme and (b) dropped requests for the simple scheme.](image)

![Fig. 5. Results from the priority scheme: (a) response times for priority scheme and (b) dropped requests for the priority scheme.](image)
queries, though this is at the expense of browse queries. In particular, the rate of increase for browse queries does not tail off as it does in the simple scheme; this is to be expected because of the priority given to purchase queries. Note that Fig. 5(b) shows that no purchase queries are dropped under the conditions of the experiment, but that the total number of queries dropped is consistent between both schemes. The fact that browse queries start to be dropped a little earlier is due to buffer sizes being small and the processor being occupied largely with purchase queries.

6. Conclusions

This paper has proposed a simple scheme for prioritising different categories of queries with the aim of increasing the performance of high priority queries at the expense of lower priority ones. The formal specification of the scheme allowed us to consider alternative approaches and refine the one chosen, as well as providing a framework for the implementation, increasing our confidence that we correctly implemented the scheme we designed. The experimental results show a marked improvement in the performance of high priority queries, though low priority queries experience reduced performance including rejected requests.

Thus, in the context of our assumptions about the behaviour of clients, the priority scheme meets our aims. It is also easily extensible: further priority levels can be defined simply by including extra buffer components and modifying the Gatekeeper to sort queries appropriately. Additionally the scheme can accommodate multiple web servers since the buffers can be accessed by n servers as easily as by one.

However, further investigation of the behaviour of clients is necessary. The simple behaviour assumed may not be realistic and if clients experience significant delays in their requests for information or have their requests rejected, then they are likely to move to a more responsive site. Conversely, clients may be tolerant of minor delays in the completion of purchase queries since this is often the final interaction with the site. Given these considerations the simple priority scheme proposed may not be satisfactory since it may lead to prospective clients leaving the site before making purchases due to delays in requests for information. It is likely that applying techniques such as weighted fair queueing (Demers et al., 1989) or threshold-based queueing (Boxma et al., 1995) will address this issue.

References


Bolexa O, Koole G, Mittrani I. A two-queue polling model with a threshold service policy. In: Third IEEE international workshop on modeling, analysis, and simulation of computer and telecommunication systems (MASCOTS ’95); 1995.


