Server Selection for Differentiated classes of users

Mohamed-Vall M. Salem

Département d'Informatique, Université de Montréal, CP 6128, Succ. Centre-Ville, Montréal, QC, H3C 3J7, Canada, salem@iro.umontreal.ca

Gregor v. Bochmann

School of Information Technology and Engineering (*SITE*), University of Ottawa, P.O. Box 450, Stn A, Ottawa, Ontario, K1N 6N5,Canada, bochmann@site.uottawa.ca

Johnny W. Wong

Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 jwwong@bcr.uwaterloo.ca

Abstract

It is expected that some applications like for instance, e-commerce systems, will be able in the future to provide different levels of service to different classes of users. Classes of service may for instance be access-oriented, performance-oriented or content-oriented. In this paper, we investigate the introduction of differentiated server selection during the phase of server selection and at independent brokerage entities and not at the server level. This has the advantage that service differentiation can be realized using a broker and a set of generic servers, thus enhancing the portability of servers. Development of servers remains generic while brokers implement sophisticated policy requirements.

1 Introduction

It is expected that some applications like for instance, e-commerce systems, will be able in the future to provide different levels of service to different classes of users. For instance, an e-commerce system may distinguish between a casual user and a registered user who is a regular customer. Some of the registered users may for example be known as heavy buyers; they may obtain the "Elite" service, while the normal registered user obtains the "Premium" service, and the casual user the "Normal" service. These different classes of service may differ in several aspects, like for instance, a shorter response time for a higher class of service. A higher class of service may also provide

facilities that are not available at the basic level, such as for instance a teleconference chat with a sales person[2].

There has been a considerable amount of work in the literature [8,4,5,3,9] on the provision of different categories of service. Most of this related work, as we will see further in this paper, requires specialized resource management schemes to be implemented at the server in order to enable differentiation between users. In this paper, we study an approach where service differentiation is delegated to a broker node. This approach is based on an architecture that we have investigated in [1]. In our architecture a "broker" is used to distribute load to a set of replicated servers for the case of one class of users. This architecture offers improved scalability. It also allows the broker to gather information about server status and client requirements, and use such information for load balancing purposes. Based on our architecture, the broker can include service differentiation as part of the server selection process. Performing differentiation during server selection has the advantage of enhancing the portability of servers. This has the advantage that service differentiation can be realized using a broker and a set of generic servers, thus enhancing the portability of servers. Development of servers remains generic while the broker may implement sophisticated policy requirements.

This paper is organized as follows. In Section 2 we look at some specific considerations that need to be taken into account for the realization of a differentiated service. Section 3 presents the key features of our underlying basic architecture. In Section 4 we extend the basic architecture in order to support service differentiation, and we propose a server selection policy for two classes of users. In Section 5 we present the evaluation of the architecture by simulation. Section 6 gives an overview of some of the related work and finally, in Section 7, we discuss some related issues and summarize our conclusion. .

2 Service Differentiation

In order to be able to implement a differentiated service, any architecture has to include mechanisms by which users are identified and classified. In this section, we consider

different alternatives for the identification and classification of users. We will discuss later in the Sections 4, and 7 of the paper how they can be supported in our architecture.

2.1 Identification of Users

The issue of user identification client-server applications has been investigated extensively. The most critical problem is the storage and retrieval of user profiles. A widely used technique in Web applications is the use of "Web Cookies" where the client software holds a small amount of state-information associated with the user. Servers send cookies to clients as part of the HTTP response headers. When a client subsequently interacts with a server, the cookies are automatically retransmitted as part of the HTTP request headers and it allows server-side to identify the user.

2.2 Classification of Users

Classification of users may be done in many ways. It depends largely on the business model used at the server side. In what follows, some of the approaches reported in the literature are discussed.

2.2.1 Static Classification

The classification of users is carried out at the server side and each user is required to register before accessing a server. During the registration phase, a user may subscribe to one of several categories of service each of which corresponds to certain privileges.

2.2.2 Dynamic Classification

The classification of users can be dynamically defined based on the identity of the user, the actual status of servers, and/or the status of the user's sessions. In such classification, the class of priority of a user may also change over time. For example in [3], sessions that are already admitted to a system have more privileges than newly arriving sessions. A user's session may therefore gain priority as the session proceeds. Another example is the classification proposed in [9] where three classes of priority are defined: low, medium and high. A new user starts at the high priority class, and his priority changes depending on the length of his session and the type of transactions he executes.

2.2.3 Classification based on request type

In this type of classification, the priority is based on request type and not on the identity of the user. This can best be illustrated by means of examples. An E-commerce site may, for example, organizes its content in a way that favors some types of transactions regardless of the identity of the user accessing the server. For example, browsing and searching transactions in an E-commerce site may both belong to the same normal class of priority while the payment transaction may be considered as the most important class.

3 Basic Architecture

Our basic architecture is depicted in Figure 1. In this architecture, scalability is achieved by server replication. We consider the use of a "broker" to distribute load to replicated servers. Our architecture allows for a flexible organization of resources used by web sites. The broker could be at the server site under the same authority as the replicated servers. This is applicable, for example, to sites with heavy load and high degree of replication. Different sites may also share the same broker. In this case, the broker could be an independent brokerage service that manages the assignment of servers for affiliated sites. A description of our architecture is as follows[1]:

Server selection is "session" based where the broker assigns a client to a specific server for a given duration of time (called the quantum). Suppose a client would like to access a given web site, say store1.com, as shown in Figure 1. If the IP address corresponding to store1.com is in the client's cache, then the request is sent directly to that IP address. Otherwise a server selection request is first sent to the broker via DNS mapping. The broker then selects a server and returns the IP address of this server, together with the quantum size, to the client. The client caches the IP address of the selected server and starts his session with the server. The cache entry will be deleted when the quantum expires. Note that the broker is only involved in the initial server selection, and the user interacts directly with the assigned server during the session.

Our architecture allows the broker to gather information about server status and client requirements, and use such information for load balancing purposes. Performance

data are collected by monitoring agents at the servers, and sent to the broker periodically.

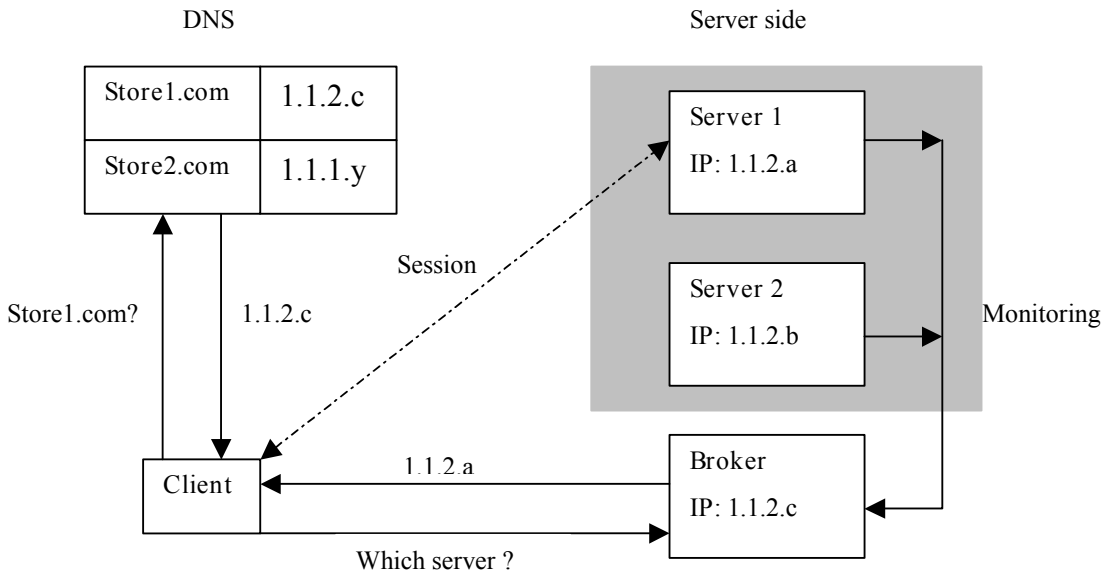

Figure 1: Basic Architecture

4 Extension to support service differentiation

In this section, we extend our basic architecture to support different classes of users. For convenience, our discussion will be focused on the static classification. The application of our extended architecture to dynamic classification and classification based on request type will be discussed later in the Section 7 of the paper.

4.1 User Classification

In a static classification model, a new user is given a default priority class by the broker. This user is then assigned to a server for a quantum of time long enough to conduct a registration. From that point on, the classification of that user will become known.

4.2 Organization of Resources at the Server-side

The realization of selection policies depends also on the organization of the available resources and hence the definition of the quotas reserved for each class of users. Servers

may be organized into sub-clusters each of which servicing a particular class of users. Or they may also be shared without any regards to user classes (i.e. users from different categories may be served at the same physical server). The definition of a quota for each category of users is a very important parameter, which may be statically defined by the system administrator based on previous knowledge of the size of each group of users for example. It may also be dynamically adjusted while the system is running, based on observed values of the incoming traffic and or the QoS delivered to users.

We will use, throughout this paper, the following notations to describe the various possible forms of organization in the case of two classes of priorities. Let Class A denotes the class of important users, and Class B denotes the class of normal users. Let C_A and C_B be the sets of servers dedicated to users of the classes A and B respectively. These sets are reserved for each class of users and may not be shared by any other class at any time. Let C_{shared} be the set of servers that may be shared among users form different classes. The resources available at the server side may be organized in different ways based on the sizes of the different sets of resources C_A, C_B and C_{shared}. For example, the following main configurations may be identified:

- Complete sharing of resources (C_A=0 and C_B=0): All the available resources are shared between the two classes of users.
- Partial sharing of resources (C_{shared}≠0, C_A≠0): A private set of resources (servers) is reserved for class A, and in this case, class B can only use the shared cluster.
- Partitioned resources (C_{shared}=0): Users from the two classes are served at two physically different sub-clusters.

4.3 Server Selection

Selection policies are often characterized by the objectives they seek to achieve. They may for instance be access-oriented, performance-oriented or content-oriented. In access-oriented objectives for example, important users are given the priority for the access to servers. The objective is then to minimize the rejection probability of the most important class of users regardless of any QoS parameters. In performance-oriented

differentiation on the other hand, the admission of users from different classes is subject to QoS verification. Users are admitted only if their specified QoS requirements may be delivered or if their presence will not affect the QoS received by active users. The objective of differentiation may also be only related to the content provided to the different customers, and in this case, users from different classes do not receive the same content while accessing the same service.

Another important factor that may characterize a policy is the level of the system at which the policy is applied and how it enforces the delivery of the quality of service committed for users. A policy may be implemented at the single server level, either by adding specific modifications to application servers or the operating system or both[4]. They may also be implemented inside independent entities like for instance "brokers" in the case of our architecture.

Enforcement of quality of service refers to the capacity of a policy to enforce the guarantees given to users. A policy may then be either preemptive or non- preemptive [8], if it may or not interrupt the service of an ongoing request or session. A preemptive policy may for example interrupt users from a lower priority class if the resources are needed for more important users. Policies may further be classified according to their flexibility in service provisioning into so-called work-conservative and non-work-conservative ones [8]. A policy is said to be work-conservative if it allows lower priority requests/sessions to obtain additional resources in the case that few higher priority requests/sessions are present. A non-work-conservative policy on the other hand will always limit lower priority requests/sessions to their initially defined quota.

We will define in the next subsection a selection policy that takes into account two classes of service and that is applicable in the context of our architecture.

4.3.1 Example: Server Selection with Performance Guaranties

The objective of this policy is to organize the admission of new users from different classes taking into account their different QoS-requirement. The class of users with the hardest requirement (important users) is then given privilege in the access to the resources at the server side. The admission of a new user (session) is restricted only if

the committed QoS for its own class of users may not be delivered at the moment or if the QoS of active users from a more important class is in danger of being violated.

The broker in charge of server selection operates in two different modes, namely the *normal* and the *differentiation* mode. A precaution threshold defines the limit between the two modes of operation. This threshold is chosen carefully to prevent, as much as possible, the violation of quality of service desired for the important class of users. In the normal mode, any load-balancing algorithm without differentiation between users may be used. The level of saturation of the system does not necessitate any differentiation since the system can handle all users with acceptable QoS. The broker enters the differentiation mode when the saturation level of the cluster of servers reaches the so-called "precaution threshold", and it starts using a differentiation policy.

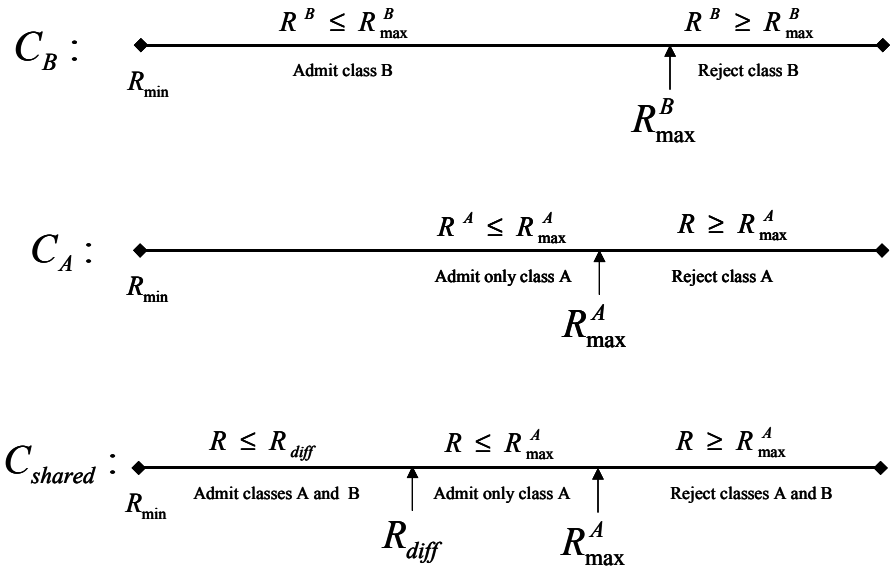

Figure 2: Differentiation between users depending on system response time

The selection policy proceeds as follows. Let $R^{(A)}_{\max}$ and $R^{(B)}_{\max}$ be respectively the thresholds for the acceptable mean response time for the classes A and B of users. Let R^A and R^B be respectively the current mean response time for the classes A and B. Let R_{diff} to be the threshold after which the system activate its differentiation mode where the access to all servers is limited to the users from the class "A". Figure 2 illustrates the different phases of the policy for the different possible organizations of

the server side, and the algorithm is given in Figure 3. The server selection policy provides the following guarantees:

- A client is admitted only if the required response time maybe delivered. This response time may be defined by the system's owners or explicitly required by customers. For example a class "A" user is rejected if the response time is longer than $R_{\max}^{(A)}$.

- There is absolute priority for active users over new users from the same class. If the response time, in the system, is beyond the acceptable thresholds, admitting any new client will only worsen the situation of active ones.

- In the case where only a shared resource (set C_{shared}) is available, the realization of the policy provides, besides the two earlier stated guarantees, an absolute priority for the important class of users. If there are enough users from class "A" requesting service, and since $R_{\max}^{(A)} \leq R_{\max}^{(B)}$, no class B users may enter the system and eventually only class A will be served.

The realization of the policy gives strong guaranties for important users but may cause a waste of resources in situations of high load with few important users and a large number of normal users. If after the response time of the cluster C_{shared} has exceeded the threshold R_{diff} and only class B is coming, the resources are not used as much as they could since the system, by precaution, rejects normal users and no important ones are coming. If however, the incoming requests are either a mixture of classes A and B or totally dominated by class A, such problem does not arise and the system operates normally favoring important users and no resources are wasted.

This selection policy may also be realized using other forms of resource organization, like for instance the use of a totally portioned cluster ($C_{shared} = 0$). In this case, the admission control at the broker enforces the admission conditions for the two classes of users on the two physically separated sub-clusters C_A and C_B. This realization of the policy has the advantage of simplicity, but the definition of the right quota for

each class of users may not be always easy to define. We will evaluate in the rest of the paper, the performance of the policy for a completely shared cluster of servers.

Class B:

If ($R^B < R^B_{\max}$ *and* $R^A < R_{diff}$)

Select the best server in (C_{shared} *and* C_B)

Else if ($R^B < R^B_{\max}$)

Select the best server in (C_B)

Else

Reject user

Class A:

If ($R^A < R^A_{\max}$)

Select the best server in (C_{shared} *and* C_A)

Else

Reject user

Figure 3: Server Selection Algorithm

5 Performance Evaluation

5.1 Simulation Model

The simulation model consists of a single broker, *N* servers and *M* concurrent clients. The *M* clients are divided into two groups of users respectively denoted by A and B. The class A represents the important users and the class B represents the normal users. For each client, a page request is submitted at the end of a "user-think time". Each page request corresponds to one or more "object requests" to be submitted to the server. We assume that at the client, object requests are submitted sequentially as required in HTTP 1.0. This is modeled as follows. When a response is received for an object request, the next object request is submitted after some processing at the client. When all the objects have been received, the page request is satisfied, and the client starts the next user think time. We further assume that objects are not cached and network delays are assumed to be negligible. As in [7], two heavy-tailed distributions, namely Pareto and Weibull, are used to model the user think time, the number of objects per page, and the processing time between requests in a single page.

Each server is modeled by two parameters. The first parameter (capacity) gives the time required by a server to process one byte of data[6]. The second parameter is the maximum waiting time for a request at the queue of a server as described in [3]. For example, if a server can serve one byte of data in 10^{-6} seconds (i.e. a capacity of 10^6 bytes/sec), and the average size of an object is 10,000 bytes, then the server can process on average 100 objects per second. In our experiments, the following configuration is used: a cluster composed of four servers, each of which has a capacity of 10^6 bytes/sec. The maximum waiting time, before a user's request times-out, is set to 2 seconds. When a user request times out, the user retries three consecutive times before giving up and aborting the session.

The load on the system is controlled using a load generator that continuously creates new users each of which enters the system for a session. The inter-arrival time between any two new users is an exponential distribution. Two levels of load are modeled during our simulations, respectively heavy and high load. The mean inter-arrival time between new users for the heavy load conditions is 0.15 seconds and is 0.20 seconds for the less heavy load conditions. The maximum acceptable response time $R^A_{\max}$ is set to 1.5 seconds during all the simulations. We will use the case of no differentiation to illustrate the difference between the two levels of loads as follows: In order for the broker to satisfy the condition $R_i \le R^A_{\max}$ for the mean response time for each server *i*, it may only admit 70.2% of the incoming sessions under the heavy load conditions and 92.86% in the (normal) high load conditions. It is worth noting that the case of $R_i \le R^A_{\max}$ represents the case where no differentiation is used between users by the broker. Each newly generated session has a probability P_A to be of class "A" and a probability $P_B = 1 - P_A$ to be of class "B". The values used during our simulations for P_A and P_B are respectively 0.1 and 0.9. The length of users' sessions is derived from an exponential distribution with a mean value of 36 pages in accordance with the results presented in [3].

The simulations are executed for a completely shared cluster of servers. The selection algorithm is implemented as follows. The broker forms two virtual groups of servers for each class of users. The membership of each group is defined based on the

response time of each server. Group A contains all servers that may serve a user from the class "A" and which verify the admission conditions for class "A"(a Class "A" user may be served at any server *i* that satisfies the condition $R_i \le R_{\max}^A$). The members of group B are all servers that verify the admission condition for class "B"(a Class "B" user may be served at any server *i* that satisfies the condition $R_i \le R_{diff}$). Both groups are initially equal and contain, as members, all the active servers under the control of the broker. Each time a performance report is received from a monitoring agent associated with a server, the broker uses the mean response time of that server to re-evaluate its membership in each group. At any instant in time, the group A contains servers with response time $R_i \le R_{\max}^A$ and group B contains servers with response time $R_i \le R_{diff}$. Upon reception of a message from a client requesting a server, the broker uses the less utilized server algorithm (LU) described in [1] to choose the less utilized server from the group that corresponds to the class of the client. If all servers in the same group have the same utilization, a random selection is used to break the tie.

5.2 Simulation Results

The performance measures of interest, in our experiments, are: (1) the probability P(c) of admission of sessions from class c, and (2) the probability R(x) that the response time for users from class "A" is less than a certain value x. P(c) is a measure that indicates the effect of the differentiated server selection on the admission of users from the different categories. It can be interpreted as follows. During the simulation, and under a specific load, P(c) is the percentage of users from class c that were admitted by the broker. R(x) is used to estimate the effect of the differentiation, between users, on the response time experienced by the important class of users.

5.2.1 Session Admission

Figure 4 shows the statistics of the admission of users from different classes at the broker level (i.e. before the beginning of their sessions). The broker achieves the main goal, which is the guarantee that all the important users are admitted to the system. As expected also, the admission of important users (class A) increases, as the access to servers is restricted for class B. Another important goal is to admit the maximum

possible number of users from class B without affecting users of class A. For example using a heavy load, a timeout of 2 seconds, and a threshold th_{diff} = 0.15 seconds, only 57% of class B are admitted to the system while the remaining 42.5 are rejected by the broker. As the admission condition is relaxed using larger values for the threshold, the admission of class B is improved. It becomes for example, as shown in the same figure, 67.4% for th_{diff} = 0.6 seconds, and 77.3% for th_{diff} = 1.05 seconds. The admission ratio for all customers when no differentiation is used equals 98%.

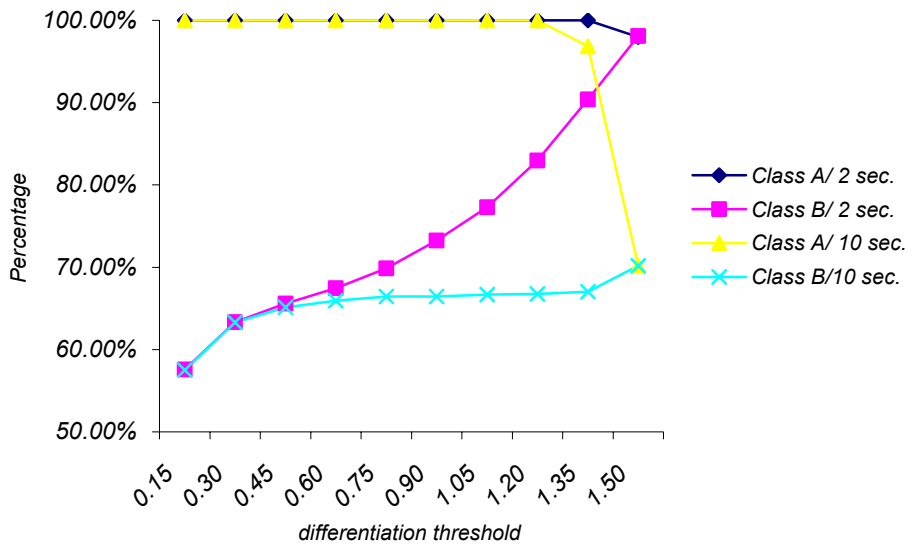

Figure 4: Statistics of admission of users in heavy load conditions

When a large timeout is used for the abortion of sessions, the results for class B as shown in the lower curve in Figure 4, are different from short timeout values for large values of the threshold. In fact when the timeout value is large, changes in the value of the threshold beyond some point (th_{diff} = 0.45 seconds in our simulation) do not have significant effects on the admission of users from class B. The difference between the two lower curves in Figure 4 for the admission of B are explained by the degree of patience of users expressed here by the timeout value. In short timeout simulations, and which corresponds to impatient users, part of the admitted users abort their sessions because of the long unacceptable waiting time at servers, which in turn results in a less loaded system. In the case of longer timeout values, users are willing to hold on for the

selected servers even if the response time is long, thus reducing any chances for the admission of new users. The apparent benefits or improvement in the admission of users from class B in the case of short timeout has a hidden side effect, which may be dramatic for some type applications like for instance Electronic Commerce application where session completion is a key parameter for the profitability of business[3].

Figure 5 shows the statistics for the completion of sessions for the users admitted by the broker in heavy load conditions. The probability that a class "A" user abort its session increases as the number of admitted users from class B increases. The probability that an important session completes normally decreases gradually as the threshold th_{diff} increases towards the point of no differentiation ($th_{diff} = R^{A}_{\max} = 1.5$ seconds). When a longer time is used (results not shown here), the admission control at the broker efficiently differentiates in favor of class "A" users who can complete normally their sessions. Our simulations showed the same results in lower load conditions but at different scales. The results are presented below for the admission of users in Figure 6 and for the completion of admitted sessions in Figure 7

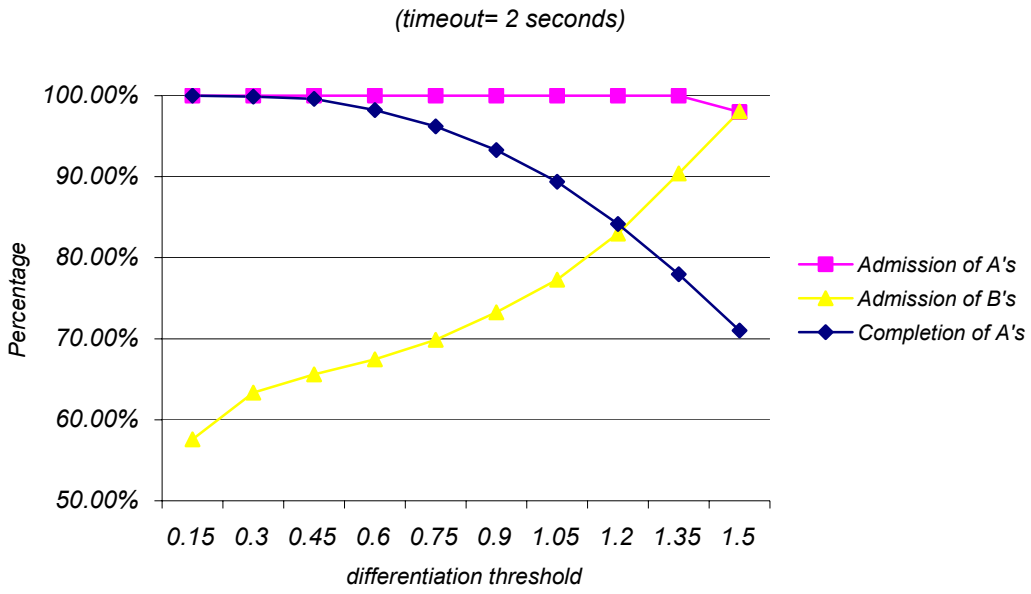

Figure 5: Statistics of the completion of sessions for class A in heavy load conditions

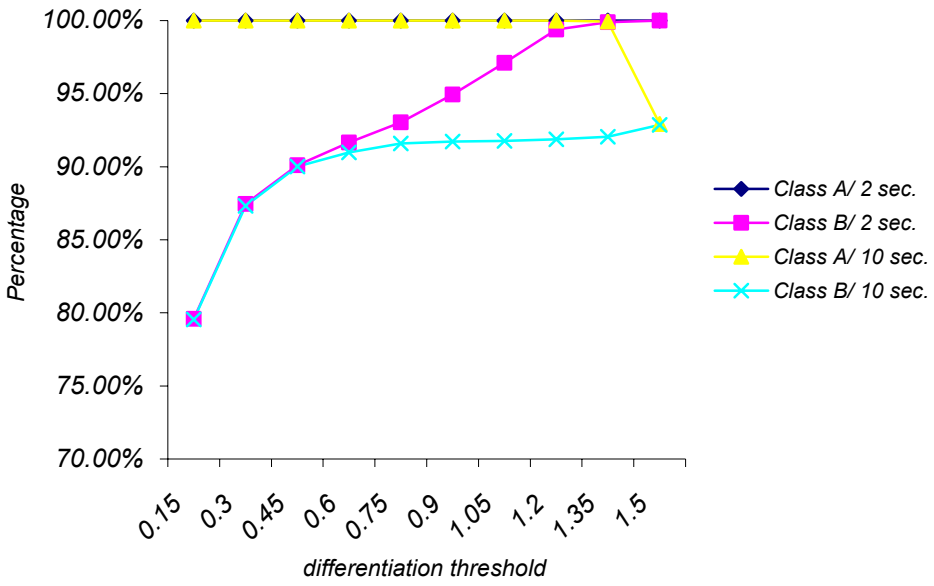

Figure 6: Statistics of admission of users in heavy load conditions

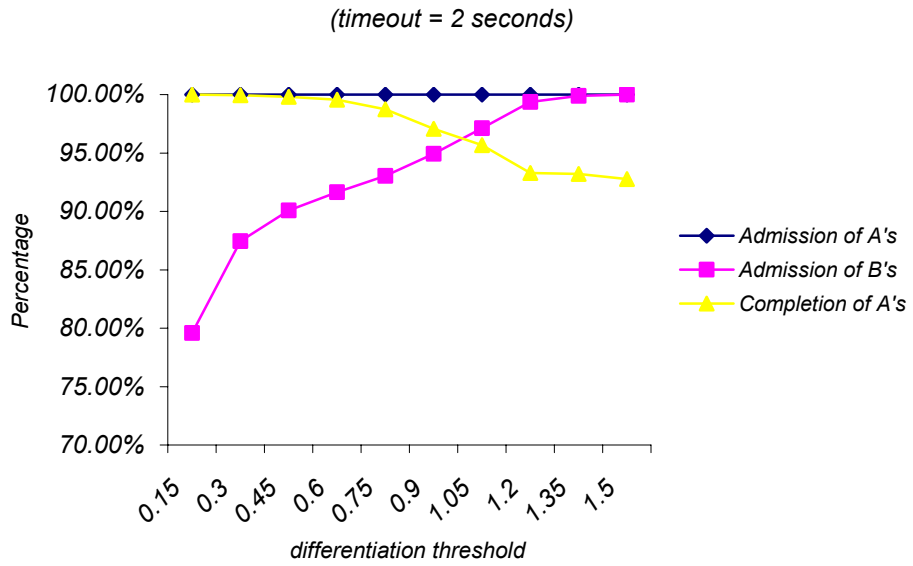

Figure 7: Statistics of the completion of sessions for class A in high load conditions

5.2.2 Response Time

The objective of differentiated services is not only to give important users the access priority to the resources available in the system. It is also very important to give

them a descent quality of service. The figures below show the effect of restricting the admission of new sessions on the QoS received by the important class of users for selected values of the differentiation threshold. The figures are organized into two groups one for high load conditions and one for heavy load conditions. In each group the figure on the left shows the results when a short timeout value is used and the figure on the right shows the results for longer timeout value. The left side of Figure 8 for example, shows the performance results achieved under heavy load conditions for different thresholds and a short timeout for requests' abortion. The percentages of requests that are completed during the simulation with a response time less than 1.6 seconds are 96%, 93%, 83%, and 67% respectively for the thresholds 0.3, 0.6, 1.05, and 1.5 seconds. The right side of the Figure 8 shows the results of simulation with longer timeout values. In this case the users are more patient and are willing to wait longer time before aborting their ongoing requests. The percentage of requests that are completed during the simulation with a response time less than 1.6 seconds are 80%, 76%, 69%, and 53%. Class "A" has very good admission results 100%, 99%, 98%, and 71 %. Figure 9 shows the results of the simulations under less heavy load condition. The figure exhibits the same general remarks mentioned in the paragraph 5.2.1 with regards to the differences in performance when the threshold is varied.

The results of the simulations showed the expected difference in performances when the threshold th_{diff} is varied from small to larger values. The smaller the value of the threshold, the better is the QoS delivered to class "A" at the price of rejecting more and more of class "B" of course. This difference is visible in the gaps between the curves that represent the performance of individual simulations in Figure 8 and Figure 9.

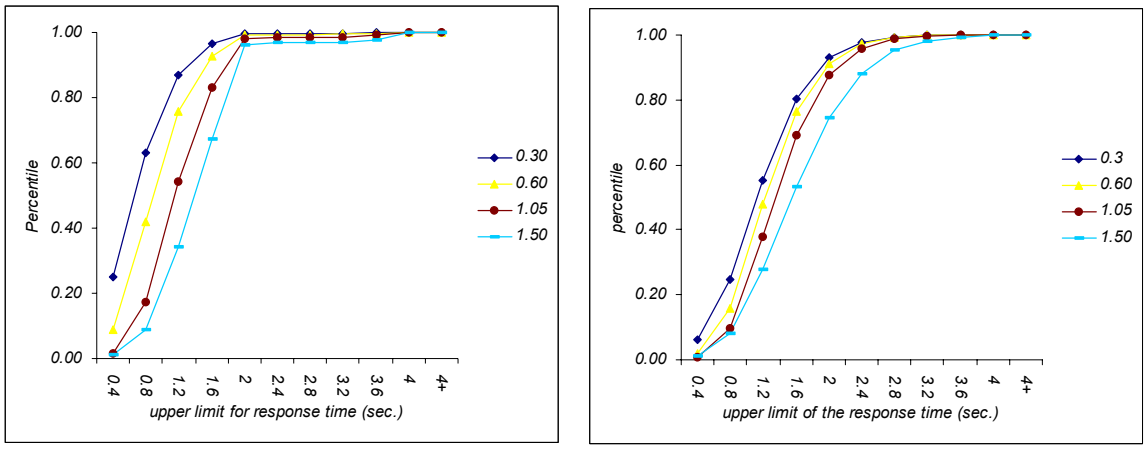

Figure 8 Cumulative distribution of the response time for class "A" in heavy load conditions using selective thresholds (left timeout =2.0 sec., right timeout =10 sec.)

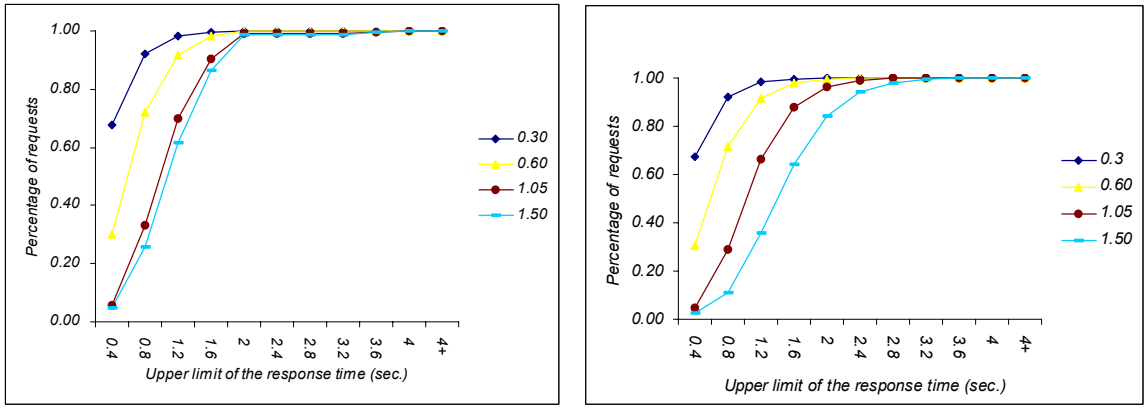

Figure 9 Cumulative distribution of the response time for class "A" in high load conditions using selective thresholds (left timeout =2.0 sec., right timeout =10 sec.)

6 Related Work

Several studies and experimentation on the provision of a differentiated service for commercial Web sites have been published in the literature. Most of these approaches are based on the modification of servers to include the differentiation between requests to the same content. In [8] Almeida et al. investigated the provision of differentiated QoS using priority-based request scheduling. They studied and experimented with two approaches, namely the user-level and kernel-level approaches. N. Vasiliou and H. Lutfiyya have also proposed a similar Web server prototype [4] that supports differentiated services at the server level. In [3], an architecture is presented for QoS in

Web-oriented systems in which, an admission control mechanism is used to improves the performance of overloaded servers and that guarantees better chances for active sessions to complete. In [9], Daniel A. Menacsé et al. presented a business oriented resource management approach that dynamically categorises customers' requests based on user profiles. The authors proposed a policy that uses three classes of priority: low, medium and high. When a new user starts a session, he enters the high priority class but after that he may remain in it or move to another class depending on the type of transactions he executes. Servers are modified to take into account the information on users priorities in the access to resources (processors, disks, etc.).

The approaches presented in [8],[4], [3], [9] necessitate the modification of generic servers to include differentiation mechanism between users. In our work, the servers are not changed, and the mechanisms are rather included in the server selection phase. The user-level differentiation mechanisms proposed in [8], and [4] can be supported in our architecture. The same remark applies for the approach used in [3], since our architecture supports the notion of session. The work proposed in [9] and the kernel-level approch in [8], all necessite the use of modified servers and operating systems to enforce service differentiation. The approach in [9] may not scale very well with a broker architecture if the class of a user changes too often.

7 Discussion and Conclusion

We have discussed in this paper the different architectural choices regarding the classification of users into priority groups and looked into the specific case of classification through registration. Our architecture can also support other approaches for classification. The dynamic classification of users can for example be delegated to the broker if the degree of importance of a user can be defined at the beginning of his session. This is applicable for example to the cases where the broker may decide based on the status of the system, the class of priority of a new user. If however the class of priority of a user depends on some changing business values that only the server can compute on the fly as described for instance in [9], the server is a more suitable place to handle the classification. The classification of customers may also be based on the delivered content or the type of transaction. In this case, the classification assumes a

model in which, requests are assigned priorities based on which documents they are requesting, not on where requests came from. This type of classification can be supported by our architecture using for example virtual sub-clusters of servers. In this case, the content offered by a server is organized into virtual sub-clusters, each of which represents a specific class and is distinguished from other classes by a different service name. The DNS system can be used to resolve all the different service names to the address of the broker and the broker identifies each class of priority by the name of the service for which a selection request is received.

In this paper, we investigated the realization of an architecture for QoS for two classes of users in client-server applications. This architecture is based on the notion of a "broker" that handles all the details of server selection. While the original design of our architecture was motivated by scalability and load sharing considerations, the specific objective of this work is, however, to extend the scalable architecture to support the provision of different classes of service. Performing differentiation during server selection has the advantage of enhancing the portability of servers. Development of servers remains generic while specialized brokers implement sophisticated policy requirements. A server selection algorithm in the context of two classes of priorities was proposed and evaluated by simulations. The simulations showed the feasibility of a differentiated service at the server selection time. This policy is based on the use of thresholds, defined by the system administrator, to control the admission of users from different categories.

There are several further extensions to this work that might be considered and more specifically what follows: (1) the use of differentiation may result in service denial for some users like for instance the lower-priority ones. An important future work will be to look into an alternative solution to the simple denial of service like for instance the use of future reservation of resources for some users. The architecture proposed in this paper is quite flexible; (2) the client-broker protocol can be extended to include quality of service negotiation on an individual basis to give users the ability to define their own requirements instead of a fixed number of classes predefined by the server side; and (3) additional scalability in the context of this architecture can be provided by the use of multiple brokers. In such context it is interesting to look into the question of negotiation

between brokers of the same service that may implement different differentiation policies on a regional basis for example.

8 References

[1] Mohamed-Vall M. Salem, J. W. Wong, and G. v. Bochmann: *"A Scalable Load-Sharing Architecture for Distributed Applications"*, in the proceeding of SoftCom'2001, Split, Croatia.

[2] Gregor v. Bochmann, Brigitte Kerhervé, Hanan Lutfiyya, Mohamed-Vall M. Salem and Haiwei Ye: *"Introducing QoS to Electronic Commerce Applications"*, in ISEC2001 held in Hong Kong, April 26-28, 2001. Springer 2001,Lecture Notes in Computer Science (LNCS) Volume 2040.

[3] Ludmila Cherkasova, and Peter Phaal: *"Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites"*. In Proceedings of Seventh International Workshop on Quality of Service, IEEE/IFIP event, London, May 31-June 4, 1999.

[4] N. Vasiliou and H. Lutfiyya, *"Providing a Differentiated Quality of Service in a World Wide Web Server",* Performance Evaluation Review, Volume 28, Number 2, pp. 22-27.

[5] N. Vasiliou and H. Lutfiyya, *"Managing a Differentiated Quality of Service in a World Wide Web Server",* in Integrated Network Management, Volume III, May 2001.

[6] V. Cardellini, M. Colajanni, P. S. Yu, *"DNS dispatching algorithms with state estimators for scalable Web-server clusters"*, World Wide Web Journal, Baltzer Science, Vol. 2, No. 3, pp. 101-113, Aug. 1999.

[7] Paul Barford and Marck Crovella, *"Generating Representative Web Workloads for Network and Server Performance Evaluation"*, in Proceeding of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 151-160, July 1998.

[8] Jussara Almeida, Mihaela Dabu, Anand Manikutty and Pei Cao, *"Providing Differentiated Levels of Service in Web Content Hosting"* , Proc. Of First Workshop on Internet Server Performance, ACM SIGMETRICS 98, June 1998.

[9] Daniel A. Menacsé, Virgilio A. F. Almeida, Rodrigo Fonseca, and Marco A. Mendes, *"Resource Management Policies for E-commerce Servers"*, *Second Workshop on Internet Server Performance*, in conjunction with ACM SIGMETRICS 99/FCRC, Atlanta, GA, May 1st, 1999.