Session-Based Selection of Servers with Different Classes of Users

Mohamed Vall O. Mohamed Salem
University of Wollongong in Dubai
Faculty of Computer Science & Engineering
Dubai, UAE
Email: salem@uow.edu.au

Abstract— **it is expected that modern applications like for instance those involving electronic commerce will be able in the future to provide different levels of service to different classes of users. This is also very appealing with the emergence of service oriented architecture in which the selection of services take into consideration quality of service. 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 selection of service provider instead of the common differentiation at servers. Performing differentiation during server selection has the advantage of enhancing the portability of servers which are kept as much as possible generic while separate entities handle specific policy requirements of applications.**

Index Terms—**server selection, differentiated services, scalability, quality of service, admission control.**

I. INTRODUCTION

The ability to provide different levels of service to different classes of clients is very appealing for applications such as electronic commerce. Clients often do not have the same QoS requirements, expectations and/or capabilities. Applications may also classify users based on specific objectives or business models. It is then expected that some distributed applications like for instance, e-commerce systems, will be able to provide different levels of service to different classes of users. For example, an e-commerce system may distinguish between a casual user and a registered user; registered users known for example as heavy buyers may receive an "Elite" service while other registered users receive the "Premium" service and the casual users the "Normal" service. These different classes of service may differ in several aspects, e.g., a shorter response time for a higher priority class. A higher priority class may also provide facilities that are not available at the basic level, such as a conversation with a sales person by teleconference [5].

There has been a considerable amount of work published in the literature [11], [7], [8], [6], [12] and [14] on the provision of different classes of service. Most of this work requires specialized resource management schemes to be implemented at the server in order to enable service differentiation. For example, the authors in [11] investigated the use of priority-based request scheduling. A Web server (Apache) is modified to include a scheduler process, responsible for deciding the order in which the requests should be handled. The scheduler restricts the maximum number of concurrent processes servicing requests of each priority. In [6], an architecture is presented for QoS in Web-oriented systems in which, an admission control mechanism is used to improve the performance of overloaded servers and to improve the chance that active sessions will complete. In [12] a business-oriented resource management approach is proposed to dynamically categorize customers' requests based on user profiles. They proposed a policy that uses three classes of priority: low, medium and high. When a new user starts a session, he or she enters the high priority class. Subsequently, he or she may remain in that class or move to another class depending on the type of transactions executed. Servers are modified to take into account the information on users' priorities in the management of resources (processors, disks, etc.). [14] investigated the provision of differentiated services for a cluster of Web servers controlled by a single front-end Web switch. They proposed in particular a centralized policy for dynamic partitioning of resources to provide service differentiation. In this proposal, A Web switch that has full control over all incoming requests enforces quality of service requirements by adjusting the quota of resources available for each class of priority.

In this paper, we present an approach whereby service differentiation is carried out, for scalability reasons, as part of the server selection process and before the beginning of a user's session with a server. For illustration purposes of the merit of this approach, we consider the architecture proposed in [4] which uses brokers or intermediate nodes to distribute load to replicated servers (providers of services). This architecture presented in generic term is in fact in line with emerging trends ([1][2] and [3]) in service oriented architectures where services need to be discovered before hand and are expected to be available from various providers and with various qualities. The architecture offers improved scalability as it delegates the functionalities of server selection and quality of service related issues to independent brokerage service. It allows the broker to gather and use information about status of servers and client requirements to better manage systems. To support service differentiation, the broker is extended to further include priority class as a key factor in the server selection process. This has the advantage that service differentiation can be realized by using an intermediate node (broker) and a set of generic servers,

doi:10.4304/jsw.7.7.1505-1513

thus enhancing the portability of servers; development of servers may then remains generic while the intermediate nodes implement sophisticated policy requirements.

This paper is organized as follows. In Section 2, we describe the key features of the underlying broker-based architecture. In Section 3 we discuss the issues that need to be addressed in order to support service differentiation. Section 4 presents a server selection policy with performance guarantees for different classes of users. In Section 5, the policy proposed in Section 4 is evaluated and analyzed using simulation and conclusions are presented in Section 6.

II. BROKER-BASED ARCHITECTURE

In this approach, a "broker" is used to distribute load between a set of replicated servers and to handle quality of service functionality. The objective is to design an architecture that is scalable to a large number of clients and that has support for the provision of quality of service. The salient features of the broker-oriented architecture are: (i) server selection is "session" based where the broker assigns a client to a specific server for a certain quantum of time, and (ii) intermediate network entities such as TCP connection routers or protocol translators, are not required in load balancing activities. Furthermore, the broker is only involved in the initial server selection, and the user interacts directly with the assigned server during the quantum. An important aspect of this architecture is the algorithm used to select servers in order to optimize the response time. Applications have the flexibility to implement selection algorithms that best suit the type of application and the targeted set of users. This 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.

The broker-based 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. In both cases, it assumed that: (1) each site (service) has its unique public name and this public name resolves to the address of its broker; and (2) each selection request sent by a user contains the name of the desired service, which allows the broker to identify the site (service) for which the request is received.

The basic architecture is depicted in Figure 1. Suppose for example that a client would like to access the site store1.com, as shown in Figure 1. If an 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 sent to the broker via DNS mapping (The URL is mapped by the DNS to the IP address of the broker). 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 that the user interacts directly with the assigned server during the session.

When the number of servers becomes large and a single broker machine is not insufficient, it may then be necessary to scale up the brokerage service by replicating the broker. In a multi-broker architecture, each broker manages the same service but interacts with a different set of servers. The various brokers communicate and cooperate in order to optimize the use of the resources available at their respective clusters. A client has first to locate one of these brokers, and then he interacts with this broker for the selection of a server. Each broker is capable of selecting a server from a remote cluster if the servers it manages are overloaded.

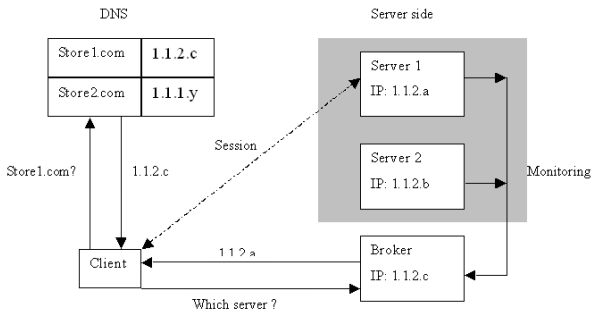

Figure 1: Basic Architecture

Server Selection Techniques

The main objective in server selection (scheduling) policies is to provide users with the maximum possible quality of service (QoS) and hence to optimize the usage of the resources available at a server side. The definition of the best QoS depends largely on the type of application and the supported policies. While some sites may be interested only in providing the best response time for their users, others may be more interested by the throughput of some particular types of transactions and/or some specific groups of customers.

There has been a considerable amount of work on the issue of server selection in traditional parallel and distributed systems [12][13][14] and more in the context of the World Wide Web environment where schedulers may not have the full control of all the traffic reaching servers for scalability reasons. The different approaches reported in the literature depend largely on the type of architecture used to provide scalable services.

Scheduling policies can be categorized into two main classes, based on whether performance information is needed or not, for the selection process. On one hand, static approaches like for instance random or simple round robin distribution of requests or sessions between servers do not use any state information in the decision process. On the other hand, dynamic approaches make

use of various types of information in order to locate the best server. Dynamic scheduling policies may use information on one or more of the following: (1) The performance of servers; (2) the performance of the path between servers and a user; (3) size and nature of user requests and (4) information from the user side. Information on servers includes, but is not limited to, metrics like the response time, the utilization of servers, the length of waiting queues at servers, the number of active connections, the number of active sessions, etc. The information on the link between a server and a client includes the Round Trip Time (RTT), the distance in number of hope, the delay, etc. Some useful information may also be collected at the user side. This includes information on performance as seen by users, and information on the behavior of users, like for instance the think time between requests. Some methods also combine various types of information on the different components of the system.

1) Selection based on server state

Server-state-based metrics are the most widely used and reported to be the most accurate and efficient in load balancing [15]. Server-base selection requires mechanisms by which performance data is collected and reported to interested entities in system. Online monitoring and remote measurement have been extensively investigated in the literature [16][17]. Even though the details of policies used in commercial products are not totally available, they are reported to use some kind of weighted round robin placement. Several policies were proposed by [18] based on information on the status of servers for the specific case of DNS-based architectures. Some of the proposed policies, like for instance the least utilized node, make use of the latest information on the utilization of servers, while some others are only based on asynchronous alarms from overloaded servers. In the first case, the server with lowest load is selected while in the later case servers are selected using some static method that excludes overloaded servers from which alarms were received.

2) Selection based on network proximity

The Internet community has extensively investigated the localization of servers based on proximity as for example in [19]. In most of these approaches, the distance is measured by tracing the route from the user to the server at the application layer or at the level of the network layer.

III. Support for Service Differentiation

In order to support effective service differentiation during server selection phase, the system architecture must include mechanisms by which users may be identified and classified. Classification allows for flexible creation of classes of priorities into which users may be grouped. The identification mechanism allows the various components of a system to recognize users for a better management of their session. The issue of user identification in client-server applications has been investigated extensively. A key requirement is the storage and retrieval of the so called user profiles. The user profile contains all the information that is pertinent for the selection of servers for a user and the management of its session. 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 [10]. Servers create and send cookies to clients as part of the HTTP response headers. The server side identifies users as it receives previously created cookies that are automatically retransmitted as part of the HTTP request headers whenever a client interacts with the server side.

Classification of Users

The classification of users into class of priority may be done in many ways. The number of classes and the membership management depend largely on the business model used at the server side. We can distinguish, in this context, between static and dynamic membership to classes of priorities depending on whether the membership of the user changes or not with time.

1) Static classification of users

In static classification of users, the class of priority of a user is defined for once and a server selection is made accordingly to that. The classification is often carried out at the server side where each user may be required to register before getting any services. During the registration phase, a user may subscribe to one of several categories of service each of which corresponds to certain privileges. The registration may also be optional and reserved for users with specific quality of service requirements while non registered users correspond to a best effort class of service.

In a broker-based architecture, a typical scenario for server selection may be as follows. For a registered user, the priority class is known (carried for example in that user's cookie) and server selection is done accordingly. A non-registered user however is first assigned by the broker to a registration server to conduct a registration and later assigned a server that corresponds to its class of priority.

2) Dynamic classification of users

The priority class of a user can be defined based on dynamically changing variable such as the load on servers and/or the status of user's sessions and may hence change over time. A user's session may therefore gain or lose priority as the session proceeds. This type of classification is application dependent and could be done in many ways. It could be for example based on the performance of servers at the arrival time of a user [3]. In this proposal, the sessions that are already admitted to a system are given more privileges than any newly arriving sessions. The status of the session of a user may also be used as criteria for the classification of users as proposed for instance in [9]. In this form of classification, the privileges of users may for example change according to some business values that are computed from the status of

their sessions. In a commercial Web site for example, the class of priority of a user can be set according to the values of products that a user has in a shopping cart. The classification of users may also be simply based on the requested content and not on the identity of the user. As an example, an E-commerce site may be organized such that browsing and searching belong to a lower priority class than payment transactions, which are considered to be more important.

Dynamic definition of classes of priorities for users can be accomplished during server selection/localization phase if the class of priority of a user can be defined at the beginning of his session. This is applicable to policies that use up-to-date information on the status of the performance of the system of servers as this information is available for server selection brokers. Classification based on content can also be supported by a broker-based architecture with proper organization of content. The same content from the same provider can be organized in virtual sub-clusters identified each by a different name (URL). Each URL may be used to represent a specific class and is distinguished from other classes by a different service name. The DNS system resolve all service names to the address of the same broker and the broker identifies each priority class by the service name for which a selection request is received.

Organization of Resources at the Server Side

To realize service differentiation, the servers may be organized into sub-clusters each of which is dedicated to servicing a particular class of users only. In addition, some of the servers may be shared by several priority classes, but the policies for admitting sessions may be different for different classes. Service differentiation can further be realized by the appropriate selection of the size for the clusters of servers dedicated for each class of users. The quantum sizes may be statically defined by the system administrator based on previous knowledge of the demand from each class. They 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.

For convenience and in order to illustrate the various possible configurations, we will consider the case of two priority classes: “A” and “B”. Class A denotes the class of important users, while Class B denotes the class of normal users. Let C_A and C_B be the sets of servers dedicated to classes A and B respectively. Also let C_{shared} be the set of servers that may be shared between the two classes. The resources at the server side may be organized in different ways depending on the size of C_A, C_B and C_{shared}. For example, the following configurations for the organization of resources may be defined:

- Complete sharing ($C_A = \Phi$ and $C_B = \Phi$): All servers are shared between the two classes of users. The admission policy controls the access of users from the different classes of priorities, but admitted users receive the same quality of service regardless of their identity.
- Partial sharing: ($C_{shared} \neq \Phi, C_B \neq \Phi$ or $C_A \neq \Phi$): A set of servers is dedicated to one of the classes A or B, while a set of shared servers exists. This type of configuration is typically used when one want to reserve a minimum set of resources (servers) for a certain class of priority.
- Partitioned resources ($C_{shared} = \Phi$): Each of the two classes is served at its own dedicated sub-cluster. This configuration is suitable for admission policies that provide different quality of services depending on the class of priority. An important class of users may not for example tolerate the same response time as a less important class of users.

Server Selection for Differentiated Services

Server selection policies with no service differentiation often strive for a better distribution of load between servers that yields a better throughput. With service differentiation however, server selection policies are often guided by other objective that need to be achieved. In this work we distinguish between objectives that are access-oriented and objectives that are performance-oriented. By access-oriented, we mean that more important users are given high priority for the access to servers. This may often be at the expense of high rejection rates for the lower priority classes. In performance-based differentiation, admission of user sessions from different classes is subject to QoS verification. Sessions are admitted only if their specified QoS requirements can be delivered and their presence will not affect the QoS received by active users from the same or more important classes. Service differentiation may also be content-based where users from different classes do not receive the same content when accessing the same service.

IV. Server Selection with Performance Guaranties

Server selection algorithm is designed to maintain a balance between access-oriented and performance based differentiation. The broker operates in two modes: normal mode and differentiation mode. In the normal mode, any load-balancing algorithm without service differentiation may be used. The saturation level 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 reaches the so-called "precaution threshold", and it starts using a differentiation policy. In the proposed server selection algorithm, the precaution threshold is defined by the value of R_{diff}, the observed mean response time above which only class A is able access servers in C_A & C_{shared}. The server selection algorithm is illustrated in

Figure 2 and Figure 3. Let $R_{\max}^{(A)}$ and $R_{\max}^{(B)}$ be respectively the acceptable mean response time for the classes A and B. Let R^A and R^B be the current mean response time for these two classes. A class A session is rejected if $R^A > R_{\max}^{(A)}$, otherwise it is admitted and the best server in ($C_A \cup C_{shared}$) is selected. Similarly, a class B session is rejected if $R^B > R_{\max}^{(B)}$, otherwise, the server selection is dependent on whether R^A is smaller than the differentiation threshold R_{diff} or not. If yes, then the best server in ($C_B \cup C_{shared}$) is selected, else the best server is selected from C_B only.

The server selection algorithm provides the following guarantees:

- A session is admitted only if the acceptable mean response time can be delivered; this response time may be defined by the system or requested by the user.
- There is absolute priority for active users over new users from the same class.
- For the case where only a shared resource (the set C_{shared}) is available, absolute priority for the important class of users is guaranteed, with the exception that class B sessions that are already active are not preempted.

It should be noted that C_A, C_B and C_{shared} can be selected to accommodate different traffic conditions. For example, a larger C_B would be appropriate for the case of few important users and a large number of normal users.

```
Class A:
    If ( R^A < R^A_max ) then
        Select the best server in ( C_shared and C_A )
    Else
        Reject user

Class B:
    If ( R^B < R^B_max ) then
        If ( R^A < R_diff )
            Select the best server in ( C_shared and C_B )
        Else
            Select the best server in ( C_B )
    Else
        Reject user
```

Figure 2: Server Selection Algorithm

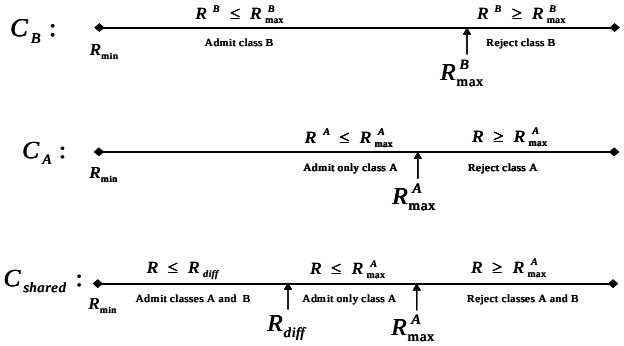

Figure 3: Example of server selection for a a cluster of servers in which resources are organized using partial sharing ($C_{shared} \neq \Phi, C_B \neq \Phi$ or $C_A \neq \Phi$) of two classes A and B of users.

V. PERFORMANCE EVALUATION

In this section, the performance of our server selection algorithm is evaluated by simulation.

Simulation Model

The simulation model consists of a single broker, N servers and M concurrent clients. The M clients are divided into two classes of users denoted by class A and class B respectively. 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. 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 object requests in a same page.

Each server is modeled by two parameters. The first parameter is capacity, measured by the time required by a server to process one byte of data [5]. For example, if a server has a capacity of 106 bytes/sec and the average size of an object is 10,000 bytes, then the server can process on average 100 objects per second. The second parameter is the maximum waiting time (denoted by W_{max}) for a request at a server as described in [3]. When a request's waiting time reaches W_{max}, a time-out occurs and the user re-submits the request; the session is aborted if the user experiences three consecutive time-outs.

The load on the system is controlled using a load generator that continuously creates new user sessions. The inter-arrival time between new user sessions follows an exponential distribution. Each newly generated session has a probability PA to be of class "A" and a probability PB =1-PA to be of class "B". The length of a user session

follows an exponential distribution with a mean value of 36 pages in accordance with the results presented in [3].

Simulation Experiments

In our simulation experiments, we consider a cluster composed of four servers, each of which has a capacity of 10^6 bytes/sec. All servers are shared by the two classes, i.e., $C_A = C_B =$□. For this special case, the selection algorithm as given in Fig. 2 is reduced to the following (we have assumed that $R^B_{\max} > R^A_{\max}$).

Class A:

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

Select the best server in C_{shared}

Else reject user

Class B:

If ($R^A < R_{diff}$)

Select the best server in C_{shared}

Else reject user

The maximum acceptable response time $R^A_{\max}$ is set to 1.5 seconds. In the above algorithm, the broker uses the less utilized server algorithm described in [1] to select the best server. If two or more servers have the same utilization, a random selection is used to break the tie. Two levels of load are simulated: heavy load and high load. The mean inter-arrival time between new sessions is 0.15 second for heavy load and 0.20 second for high load.1. For each newly generated session. PA, the probability that the session is of class A is set to 0.1. This implies that PB = 0.9.

Simulation Results

In our investigation, the performance measures of interest are the followings:

- Admission of users: The probability P(x) that a session from class x is admitted (x = A or B); P(x) is a measure that indicates the effect of the differentiation at server selection time on the admission of users from different categories with different level of importance to a service provider.
- Session completion: The probability PC(x) measures the chances that a user belonging to class x will complete his or her session properly; i.e. the probability that a session is not aborted before completion.
- Response time experience by users: the probability R(x) that the response time for users from class A is less than x. R(x) is used to estimate the impact of the selection process on the quality of service experienced by important users estimated here using the response time.

[1] The results in section 5.5.3 show that mean inter-arrival times of 0.15 and 0.2 second correspond to admission percentages of 70% and 93% respectively under the condition of no service differentiation.

1) Session Admission

Figure 4 shows the results for P (A) and P (B) for different values of R_{diff}. Two values of W_{max} are considered; they are 2 seconds and 10 seconds respectively. We observe that the main goal of service differentiation is achieved, namely to guarantee that all the important users are admitted to the system. However, if R_{diff} is too small, the guarantee to class A may result in a significant penalty to class B. For example, when W_{max} = 2 seconds, only 57% of class B sessions are admitted to the system at R_{diff} = 0.15 seconds. The impact on P(B) is more severe when W_{max} = 10 seconds. For both values of W_{max}, one may relax the admission condition for class B by using larger values of R_{diff} without affecting the value of P(A). For example, for W_{max} = 2 seconds and R_{diff} =1.05, P(A) is still close to 100% while P(B) has increased to 77%. Finally, for case of no service differentiation (or $R^A_{\max}$ = 1.5 second), P(A) = P(B) = 98% and 70% when W_{max} = 2 seconds and 10 seconds respectively.

The difference in P(B) between W_{max} = 2 seconds and W_{max} = 10 seconds can be explained as follows. The case of W_{max} = 2 seconds corresponds to short time-out situations, or impatient users. Some of these users abort their sessions, which in turn results in a less loaded system. This would tend to increase the chance that a new session is accepted, thus a larger P(B). The apparent benefits or improvement in P(B) for the case of short timeout has a hidden side effect, which may be dramatic for applications like e-commerce where session completion is a key parameter for profitability [3].

Figure 5 shows the results for session completion for class A (PC (A)) as a function of R_{diff} (the threshold for response time) under heavy load and for W_{max} = 2 seconds and 10 seconds respectively. The corresponding results for admission probabilities P(A) and P(B), as are shown in Figure 4. We observe that PC(A), the completion probability of class A sessions, decreases with R_{diff} due to the fact that more class B sessions are admitted for larger R_{diff} resulting in longer delay experienced by class A requests. Simulation results have also been obtained for the case of high load conditions. These results are similar to those for heavy load, but at different scales. They are presented in Figure 6 and Figure 7 where session acceptance probability and session completion probability are plotted against R_{diff}.

These experiments are showing the efficiency of using the threshold to control the balance of concurrent sessions from different categories of users.

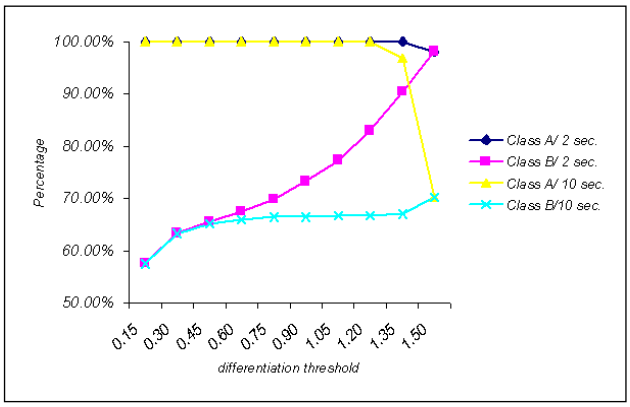

Figure 4: Probability of acceptance of user sessions (heavy load conditions)

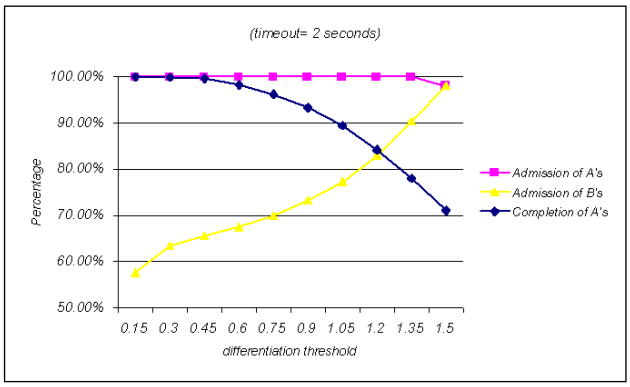

Figure 5: Probability for session completion for Class A (heavy load conditions)

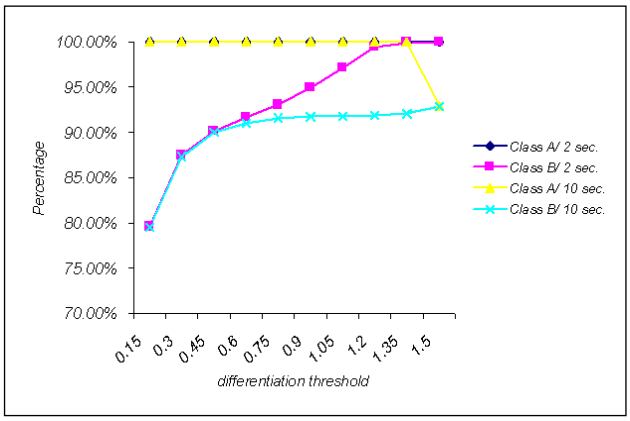

Figure 6: Probability of acceptance of user sessions (high load conditions)

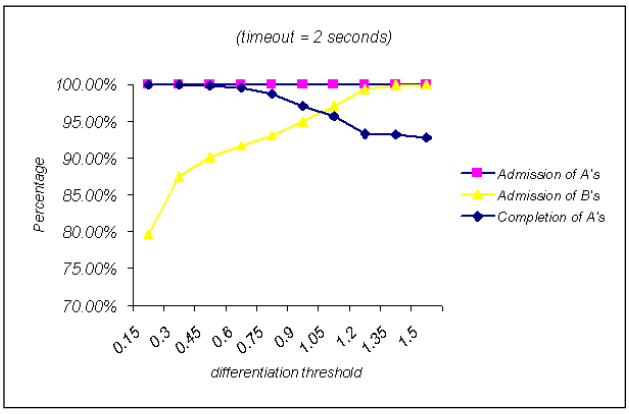

Figure 7: Probability for session completion for Class A (high load conditions)

2) Response Time

One of the main objectives of service differentiation is to provide important users with better guarantees on the quality of service in addition to the access priority to resources in the system. In Figure 8 and Figure 9 the effect of restricting the admission of new sessions on R(x) for class A is shown for different values of R_{diff} when the load condition is heavy.

For a short time out (W_{max}=2 seconds), the percentages of requests that are completed with a response time less than 1.6 seconds are 96%, 93%, 83%, and 67% respectively for R_{diff} equal to 0.3, 0.6, 1.05, and 1.5 seconds.

When a longer timeout interval is used (W_{max}=10 seconds), the users are more patient and are willing to wait longer time before aborting their ongoing requests. As a result, the percentage of requests which are completed with a response time less than 1.6 seconds are smaller, and are equal to 80%, 76%, 69%, and 53%. Figure 10 and Figure 11 show similar results of the case of high load conditions.

These results exhibit the same general trend as that for the case of heavy load. Our simulation results show the expected difference in performances when the threshold R_{diff} is varied. The smaller the value of R_{diff}, the better is the QoS delivered to class A at the expense of rejecting more of class B sessions and more class A sessions being aborted.

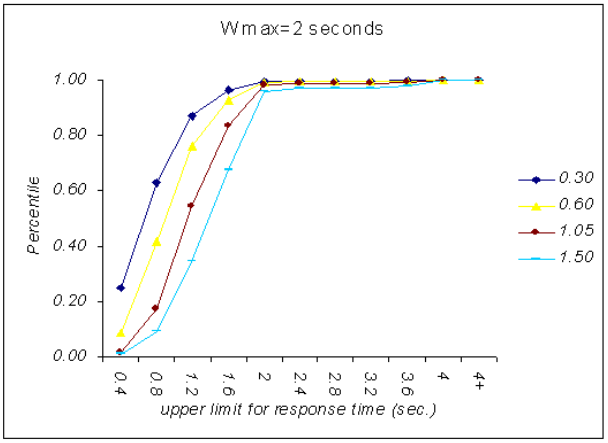

Figure 8: R(x) for class A (heavy load conditions & W_{max}=2 seconds)

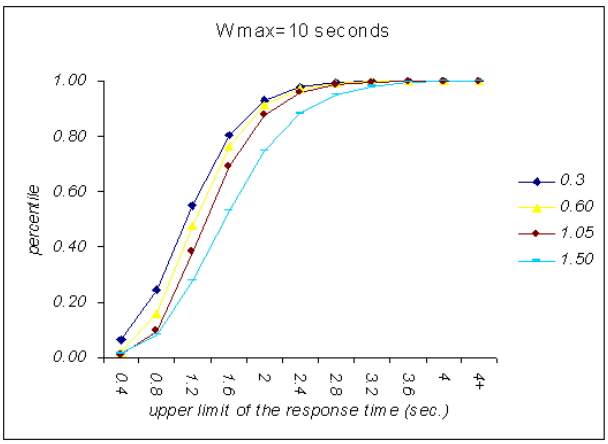

Figure 9: R(x) for class A (heavy load conditions & W_{max}=10 seconds)

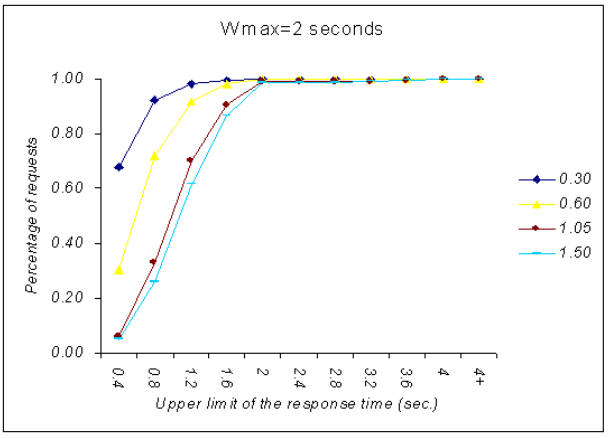

Figure 10: R(x) for class A (high load conditions & W_{max}=2 seconds)

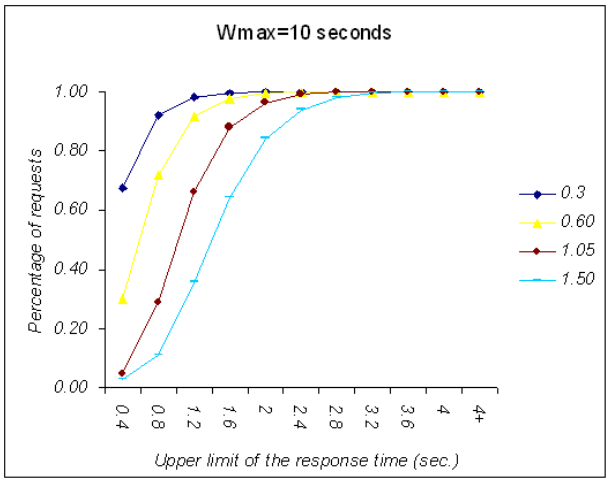

Figure 11: R(x) for class A (high load conditions & W_{max}=10 seconds)

VI. CONCLUSIONS

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.

In this study we looked at two keys performance measures to assess the effectiveness of this approach. For casual or regular users, we looked at the probability that user is admitted while for the important users, that are admitted any way, we looked at the quality of service they experience; we used, in this study, as measure of the quality of service, the probability that the response time for important users is less than a pre-defined threshold.

The results, presented in more details in the paper, indicated the effectiveness of implementing service differentiation at server selection time. This efficiency was evaluated based on statistics on the probability of admission of users' sessions, their chances of completing these sessions and the effect of differentiating between classes of users on the response time of servers. It is expected that administrators of servers will be able to tune these factors, based on their established policies, to define the capacity needed at the server side and to enforce quality of service requirements in a scalable way at the time of admitting users.

There are however several interesting extensions to this work that worth consideration. Service differentiation may sometime result in service denial for some users like those belonging to lower-priority classes. An important extension is to investigate alternative solutions such as the use of reservation of resources for some user classes. Instead of simple rejection, a user may for instance receive an offer to come back in a certain period of time where he will be guaranteed admission. The protocol governing interactions between users and the broker can be extended to include quality of service consideration on an individual basis to give users the ability to define their

own requirements instead of a fixed number of classes predefined at the server side.

ACKNOWLEDGMENT

The author wishes to acknowledge the contribution and feedback from Prof. Johnny Wong and Prof. Gregor von Bochmann at the time when this work was initiated.

REFERENCES

[1] Menasc´e, D.A., Ruan, H., Gomaa, H.: QoS management in service oriented architectures. Perform. Eval. 7-8(64), 646–663 (Aug 2007)

[2] Menasc´e, D.A., Ewing, J.M., Gomaa, H., Malek, S., Sousa, J.P.: A framework for utility-based service oriented design in sassy. In: WOSP/SIPEW ’10 (2010)

[3] Cardellini, V., Iannucci, S.: Designing a broker for QoS-driven runtime adaptation of SOA applications. In: IEEE ICWS 2010 (Jul 2010)

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] D. Kristol and L. Montulli: “HTTP State Management Mechanism”, RFC 2109 (February 1997).

[14] V. Cardellini, E. Casalicchio, M. Colajanni and M. Mambelli: “Web Switch Support for Differentiated Services,” ACM Performance Evaluation Review, Vol. 29, No. 2, pp. 14-19, September 2001.

[15] S. Bhattacharjee, M. H. Ammar, E.W. Zegura, V. Shah, Z. Fei: ”Application Layer Anycasting,” in Proceedings of INFOCOM 97, 1997

[16] B. Schroeder, “On-Line Monitoring: A tutorial,” IEEE Computer, Vol. 28, pp. 72-78, June 1995.

[17] F. Lange, R. Kroeger, and M. Gergeleit, “ Jewel: Design and Implementation of a Distributed Measurement System,” IEEE Transactions on Parallel and Distributed Systems, vol. 3, pp 657-671, November 1992.

[18] M. Colajanni, P.S. Yu, D.M. Dias, “Analysis of task assignment policies in scalable distributed Web-server systems,” IEEE Trans. on Parallel and Distributed Systems, Vol. 9, No. 6, June 1998.

[19] Z. Fei , S. Bhattacharjee, E.W. Zegura, M. H. Ammar: ”A Novel Server Selection Techniques for Improving the Response Time of a Replicated Service,” in Proceedings of INFOCOM 98, 1998.

[20] Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Trans. Web 1(1), 1–26 (2007)

Mohamed Vall Ould Mohamed Salem is currently working with the University of Wollongong in Dubai. He received a Master of Science and PhD in Computer Science from the University of Montreal (Canada). While a postgraduate student, he received the IBM Centre for Advanced Studies Ph.D. fellowship for three consecutive years and the "Bourse d’excellence de la Francophonie" from the Canadian International Development Agency.

His areas of interest and expertise are in both technical and non-technical aspects of computing including software engineering, performance analysis and scalability, innovation and ethics in information technologies.

Dr Mohamed Salem is currently the Dean of the Faculty of Computer Science and Engineering, University of Wollongong in Dubai. Previously he worked as a software engineer for several years, undertaking collaborative research with the IBM Center of Advanced Studies (Toronto) and HP Protocol Test Center (Montreal), and worked for some time as a product analyst for a start-up software company in Canada.