A business-oriented load dispatching framework for online auction sites

Daniel A. Menascé
Department of Computer Science
George Mason University
Fairfax, VA 22030, USA
menasce@cs.gmu.edu

Vasudeva Akula
School of Information Technology & Eng.
George Mason University
Fairfax, VA 22030, USA
vakula@gmu.edu

Abstract

Online auction sites have unique workloads and user behavior characteristics that do not exist in other e-commerce sites. Earlier studies by the authors identified i) significant changes in the workload depending on time of day and ii) the presence of heavy-tailed distributions involving bidder and seller behavior. Other studies indicate that a small fraction or power users, among millions of registered users, contribute to the majority of the site's revenue. Poor quality of service to power users may imply in loss of business and cause significant loss of revenue to an auction site. This problem could be mitigated by dedicating resources in an exclusive manner to this small group of users in order to improve the quality of service they receive. However, this approach can lead to i) under utilization of these dedicated resources and ii) overloading of resources allocated to regular users when the load from power users is low. In this paper, we propose a scheme whereby resources are primarily dedicated to power users, but regular users can take advantage of these dedicated resources when spare capacity is available. This paper provides a business-oriented *framework for dispatching requests to the various servers of an online auction site. Our approach uses a controller that can dynamically shift the load to different types of servers as the workload changes. Experimental evaluation showed, among other things, that the total number of bids processed using a dynamic controller to allocate resources can be improved under heavy load conditions compared to using a load balancing technique that does not differentiate among users.*

1. Introduction

Online auction sites have very specific workload characteristics as shown in a very detailed study that used data from a major online auction site conducted by the authors [3]. One of the numerous conclusions of that work, is the presence of heavy-tailed distributions in many aspects of an online auction's user behavior and bidding activity. For example, a very large percentage of auctions have a relatively low number of bids and bidders and a very small but non-negligible percentage of auctions have a high number of bids and bidders. A large percentage of auctions have a low closing price and a very small but non-negligible percentage of auctions have a large closing price.

The findings of that study allowed us to i) create an experimental testbed for online auction sites that extends RUBiS: Rice University Bidding System (see http://rubis.objectweb.org/)—a benchmark designed by Rice University—with a workload generator that is more realistic and follows the distributions observed in a major online auction site, ii) design resource allocation schemes that exploit the observed characteristics of the workload and are based on bidding activity in order to increase the revenue of the online auction site. As an example, on a previous study, the authors designed and evaluated novel cache placement and replacement policies for server-side caches in online auctions [14, 15]. In another example, the authors proposed and evaluated a closing time rescheduling algorithm aimed at smoothing out the load peaks observed at real online auction sites as the closing time for an auction approaches [16].

There are periods of time during the day in which the activity on an auction site is more intense than in others. Auction sites require users to authenticate themselves before performing revenue generating transactions such as bidding and creating auctions. Once the user is authenticated, historical activity about that user can be used to provide differentiated service to that customer. Typically, a small percentage of users contribute to a large percentage of revenue for an auction site [6]. We use some of these facts to develop user profiles with the information necessary to provide differentiated services to address some of the performance issues during peak periods on auction sites.

We present business-oriented resource allocation poli-

cies for online auctions as a mechanism to address some of the performance issues during peak-periods on online auctions, i.e., when the site cannot process all the requests in a timely manner. This technique takes user profiles and business-oriented metrics into consideration to provide differentiated service to the users on the site, instead of treating all requests with the same priority.

More specifically, our techniques are applied to the dispatching of requests to the servers of the auction site. Typical sites use some kind of load balancer that implements policies aimed at balancing the load and improving response time but are oblivious of user characteristics. Here we propose new load dispatching policies that allocate resources, i.e., servers, based on user priorities. In this paper we divide the users of an online auction site into two categories: *priority* users, who are responsible for submitting a significant number of bids, and *regular* users who tend to conduct many online searches but do not bid as often as priority users. Since the revenue of an auction site is directly related to the bidding activity of its users, it makes sense to give higher priority in terms of resource allocation to users that are responsible for generating more bids. An online auction site should provide a lower response time to priority users in order to reduce their abandonment rate. This is achieved at the expense of the performance seen by regular users.

A straightforward way of guaranteeing that priority users receive the best possible performance is to dedicate ample resources to them. This is an expensive and wasteful proposition. A more efficient approach is to set aside resources that are primarily dedicated to priority users but that can be shared with regular users as the load of priority users decreases. In this paper we describe and evaluate a load dispatcher that makes dynamic decisions based on the relative loads of these categories of users.

We measure the effect of our dynamic load control policies using business-oriented metrics such as the total number of bids generated and compare traditional performance metrics such as average response time and CPU utilizations to analyze their effect when using the new policies.

The main contribution of this paper is a *business-oriented* approach to allocating servers to incoming requests of an auction site. Traditional approaches to load dispatching are aimed at optimizing performance without considering the nature of the requests and their business value. The approach considered in this paper is aimed at improving the revenue of an auction site by providing better performance to groups of users that have higher business value at the expense of other less important users. More specifically, the contributions of this paper are:

- the design of a dynamic load dispatcher for application servers that employs business-oriented criteria to route requests to the various application servers,
- an implementation of a testbed for auction sites that uses realistic workload generators and the implementation of the dynamic load dispatcher in the testbed, and
- an experimental evaluation of the performance of the dynamic dispatcher and a comparison with that of a traditional load balancer; it is shown that the dynamic controller can increase the number of bids processed by the auction site even at high loads.

The rest of the paper is organized as follows. Section 2 provides the background and basic concepts needed in the following sections. Section 3 presents an overview of the characteristics of the workloads generated by the two types of users, priority and regular. Section 4 describes the approach used by the dynamic load dispatcher. The next section describes the experimental testbed used to evaluate the controller. Results of the experiments are discussed in section 6. The next section discusses related work and section 7 presents some concluding remarks.

2. Background

The typical architecture of an auction site is multi-tiered and is composed of three layers (see Fig. 1). The first layer comprises web servers that handle the incoming HTTP requests and serve static HTML pages. A load balancer dispatches incoming requests from the Internet to one of the various Web servers. Most pages served by an auction site are dynamically generated by an application server, which implements the site's business logic. A dynamic application-level load balancer, the subject of this paper's analysis, dispatches requests to the various application servers. An application server may need to access persistent data stored in a back-end database server. An example is the processing of a request to view all bids for a given auction. The bid information comes from the database. The application server then generates an HTML page with the bid information. This page is passed back to the web server, which sends the page back to the browser.

3. Workload Description

The workload generated by priority and regular users is described using a Customer Behavior Model Graph (CBMG), as defined by Menascé et al. [19, 17], for each category of users. Each node of a CBMG represents a state in which a user may be during a session. Nodes of a graph are connected by directed arcs that indicate the possible transitions between states. Arcs are labeled

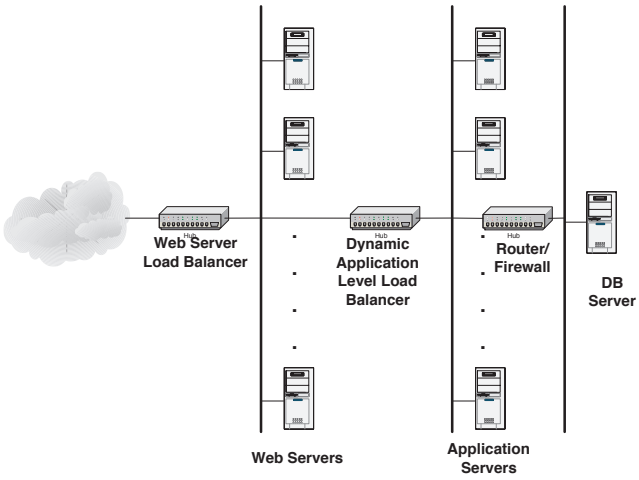

Figure 1. Architecture of an Auction Site.

with the probability that a transition between states occurs.

Figure 2 describes the CBMG for priority users. This CBMG and the one for regular users (see Fig. 3) are a simplified version of the ones actually used in the experiments. The figures only show the most important states, and in particular, those that differentiate the navigational behavior of priority and regular users. It is important to note also that in each of these CBMGs there is an additional state, the Exit state, not shown in the pictures to make them easier to read.

All the states whose sum of the probabilities of the outgoing transitions does not add to one have a transition to the Exit state with a probability equal to 1 - the sum of the other transition probabilities. For example, in Fig. 2 there is a transition to the Exit state from the Home Page with probability equal to $1 - (0.15 + 0.7 + 0.1) = 0.05$. As it can be seen from this figure, priority users start to bid almost immediately after login in to the About Me page (this is the equivalent of a personalized *my auctions* page that allows users to track the auctions they are following) without spending much time searching for items to bid.

Regular users on the other hand, spend most of their time searching and browsing for items to bid as seen in the CBMG of Fig. 3.

4. The Dynamic Load Dispatcher

In order to describe the operation of the dynamic load dispatcher, assume that there are N application servers, of which P are primarily dedicated to priority users and the remaining $N - P$ are dedicated to regular users. The servers dedicated to priority users are called *priority* servers and the remaining ones are called *regular* servers. We consider and

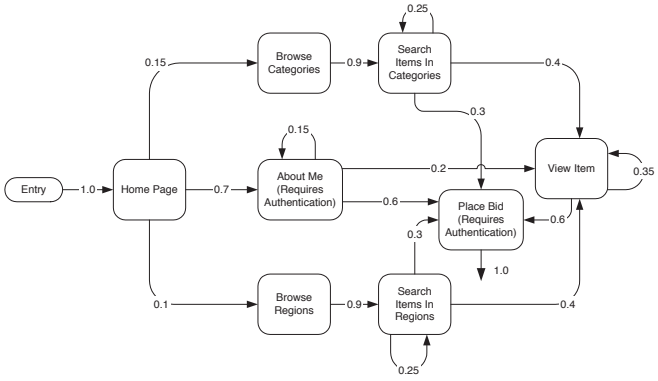

Figure 2. CBMG for Priority Users

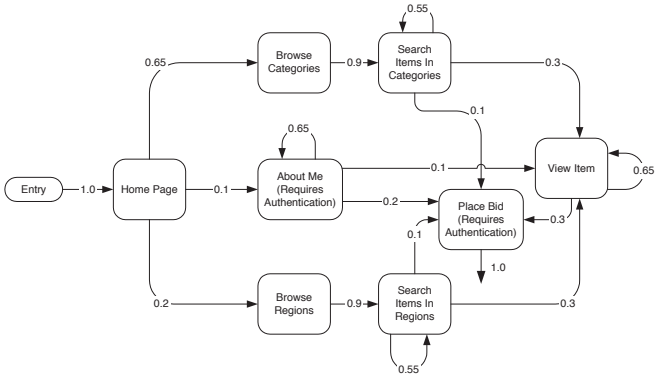

Figure 3. CBMG for Regular Users

compare two dispatching policies with respect to sending requests to the application layer servers. The first and traditional approach is pure load balancing (LB) in which the application level load balancer sends requests to the application servers in a way that attempts to give each of them $1/N$ of the total load, irrespective of the type of user. The second approach, the one proposed in this paper, is priority based (denoted PR in what follows) and dispatches requests to servers according to the rules described in what follows.

- All requests from sessions started by priority users are evenly distributed to the P priority servers.
- A dynamically adjusted fraction f of the requests generated from regular users is evenly distributed among the priority servers and a fraction $1 - f$ of these requests is evenly distributed among regular servers.

The utilization U_p and U_r of the priority and regular servers can be computed using basic principles of queuing theory as follows

$$U_p = \frac{\lambda_p \times \bar{x}_p + f \times \lambda_r \times \bar{x}_r}{P} \quad (1)$$

and

$$U_r = \frac{(1-f) \times \lambda_r \times \bar{x}_r}{N-P} \quad (2)$$

where λ_p and λ_r are, respectively, the overall arrival rates of priority and regular requests to the online auction site, $\bar{x}_p$ and $\bar{x}_r$ are, respectively, the service demands of requests of priority and regular users.

Suppose that one may want to keep the utilization of the priority servers capped at $U_p^{\max}$ (e.g., 40%) in order to provide good response time to priority users while keeping the utilization of regular servers at a minimum of $U_r^{\min}$ (e.g., 10%) to avoid shifting too much load to priority servers. Then, we can use Eqs. (1) and (2) to compute an upper bound on the value of f that will guarantee this as

$$f \leq \min\left[\max\left(\frac{U_p^{\max} \times P - \lambda_p \times \bar{x}_p}{\lambda_r \times \bar{x}_r}, 0\right), \max\left(1 - \frac{U_r^{\min} \times (N-P)}{\lambda_r \times \bar{x}_r}, 0\right), 1\right]. \quad (3)$$

For example, when $N = 20$, $P = 5$, $\lambda_p = 40$ requests/sec, $\lambda_r = 400$ requests/sec, $\bar{x}_p = 1.5$ msec, $\bar{x}_r = 5$ msec, $U_p^{\max}$ = 40% and $U_r^{\min}$ = 10%, the fraction f of regular requests that should be shifted to priority servers cannot exceed 25%.

The dynamic load controller proposed in this paper adjusts the fraction f as the utilization of the servers varies due to variations in the arrival rates λ_p and λ_r. For example, f is reduced when the utilization of priority servers is very high. But as this utilization decreases and the utilization of regular servers increases, f can be increased to shift some load from regular to priority servers. Note that the goal is to provide better service to priority users but at the same time provide good service to regular users if possible.

Table 1 shows how f varies as a function of the utilization of priority and regular servers. The utilization of the priority and regular application servers are divided into five categories: Very High ($\geq 75\%$), High ($< 75\%$ and $\geq 60\%$), Medium (< 60 and $\geq 45\%$), Low ($< 45\%$ and $\geq 30\%$), and Very Low ($< 30\%$).

The utilization of each application server is measured at regular time intervals (5 minutes in our experiments) and at each measurement interval the value of f is changed according to Table 1. The table shows that for each combination of the utilization values of the priority and regular application servers, the fraction f is either decreased or increased by a percentage factor "high" or "low" or left unchanged. The values in the table are created in such a way that the load on the priority server is one level below that of the regular server. In our experiments, we used the values 15% for "high" and 5% for "low". The values of "high" and "low" were determined after some initial experimentation. However, one could use autonomic computing techniques to dynamically adjust these values. So, for example, when both utilizations are Very High, the value of f is decreased by 15%, decreasing the amount of traffic routed from the regular to the priority server. If, however, the utilization of the priority server is Medium and that of the regular server is Very High, the fraction f is increased by 15% increasing the amount of load shifted from the regular to the priority servers.

The purpose of the controller algorithm encoded in Table 1 is to have the site spend most of the time in the cells indicated by "no change". In these cases, the utilization range of the priority server is always one level below that of the regular server (e.g., High for the priority server and Very High for the regular server). The experiments reported in Section 6 show that this is achieved in most cases.

In general, the table shows that when the priority server has a high utilization, the amount of traffic shifted from the regular to the priority servers will be continuously reduced. But, as the utilization of the priority server decreases, the rate of decrease of f decreases and when the utilization of the priority server is low and the load on the regular server is high, then more load is shifted to the priority server.

5. The Experimental Testbed

Figure 4 depicts the testbed we built to evaluate the ideas developed in this paper. Three client machines run client emulators that mimic user sessions following the CBMGs described in Section 3. These clients submit their requests to an auction site connected to them through a LAN. The auction site has a three-tiered architecture with one web server, two application servers (one priority and one regular), and a database server. The web server runs Apache and uses `mod_jk` to communicate with the Tomcat application servers, which run a servlets version of RUBiS. The database server runs MySQL.

The two application servers run on identical hardware: a 666 MHz Pentium 3 with 512 MB of RAM. The Web server runs on a 2 GHz Pentium CPU and the database server on a 2.4 GHz Pentium 4 machine. We decided to use identical and comparatively low end hardware for the application servers to ensure that application servers became the bottlenecks in order to evaluate various load balancing policies at these servers. This might be close to a realistic scenario since most of the content on auction sites is generated dynamically. The dynamic controller is implemented as a shell script that measures the CPU utilization on the two application servers every five minutes. The script dynamically up-

Priority Server Utilization ↓	Regular Server Utilization →				
	Very High	High	Medium	Low	Very Low
Very High	-high	-high	-high	-high	-high
High	no change	-low	-high	-high	-high
Medium	+high	no change	-low	-high	-high
Low	+high	+high	no change	-low	-high
Very Low	+high	+high	+high	no change	no change

Table 1. Variation of f as a function of the utilization of priority and regular servers.

dates the `JkMountFile`, a file containing multiple mappings from a context to a Tomcat worker, on the web server to indicate which load balance configuration should be used for regular requests depending on the current load distribution and CPU utilization.

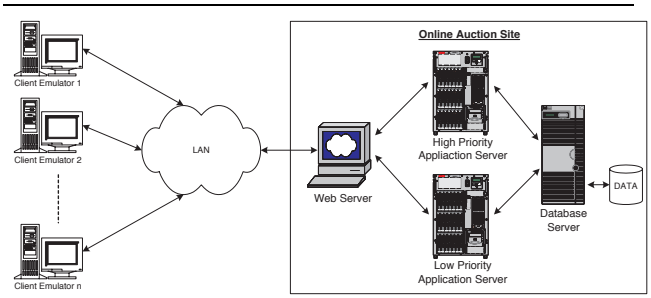

Figure 4. Experimental testbed

Our testbed auction site is based on RUBiS but we modified its client emulator to generate unique URLs with the keywords "regular" and "priority" as part of the HTTP address to allow the Web server to distinguish between requests generated by regular and priority users. The client emulators generate 20% of requests as coming from priority users following the CBMG of Fig. 2 and 80% as coming from regular users who generate requests following the CBMG of Fig. 3. We configured the workload in such a way that the number of bids generated by 20% of priority users is about the same as the number of bids generated by the other 80% of regular users. Note that, in a real world scenario, since a majority of auction site users login to track their auctions, the site can identify users and dynamically encode the URL to include the priority of the users using their profiles stored in the database. User priorities can be identified based on various business-oriented policies such as current participation in an active auction, value of their current auctions, value of their historical auctions, remaining time to close for their auctions, their winning percentage, etc. The algorithm to compute user priorities is out of the scope of this paper and it could consider several of these factors and might even classify a given user as a priority user for one season but as a regular user for another season based on the user's historical behavior. We chose to simplify this logic and statically encoded the URL with two priority levels.

Table 2 shows the distribution of ten of the most relevant types of requests generated by each type of user during the experiments. While there are other types of requests generated by the workload generator used in the experiments, those in Table 2 are the ones that are more relevant to the discussion in this paper because of the significant difference in frequency of occurrence between priority and regular users. The most frequent requests for regular users are Search Items in Categories (32.79%) and View Items (33.03%). Note that Place Bid has a very low frequency (5.43%) for these users. Consider now priority users. The highest frequency transaction (41.21%) is login to the About Me page followed by Place Bid (30.78%).

We increase the number of client emulators from 100 to 500 on each of the three different machines resulting in 300 to 1500 overall parallel clients. Each experiment has a 2-minute warm up phase, a 15-minute steady state and then a 2-minute cool down phase. We used `sar` to capture system performance metrics on the various servers used in the system. Each experiment was repeated 15 times to compute average metrics and 95% confidence intervals. The confidence intervals are very tight as shown in the graphs discussed in the next section.

6. Experimental Results

The experiments are set up so that a request is rejected if it takes longer than 5 seconds to respond. Note that by adding the network time, it could take over 8 seconds, which is the average time users can tolerate for a response from an e-commerce site [19]. This is done to mimic typical abandonment rates in e-commerce site.

We look first at the total number of bids successfully placed by both regular and priority users as the load increases (see Fig. 5). Two load distribution scenarios are

Transaction Name	Regular Users	Priority Users	% Regular Users	% Priority Users
Home	5439	486	6.43	3.14
Browse Categories	7549	398	8.92	2.57
Browse Regions	2705	153	3.20	0.99
Search Items In Categories	27751	476	32.79	3.07
Search Items In Regions	2569	151	3.04	0.98
View Items	27955	1725	33.03	11.14
View Bid History	1706	703	2.02	4.54
View User Info	1500	244	1.77	1.58
Place Bid	4597	4765	5.43	30.78
About Me	2861	6379	3.38	41.21
Total	84632	15480	100.00	100.00

Table 2. Distribution of transaction type per user category.

considered: Load Balance (LB) in which both application servers receive roughly the same load and Priority Based (PR) in which the load controller implements the dynamic algorithm described in Section 4. The number of clients varies from 300 (light load) to 1,500 (heavy load). At light loads, both schemes are identical. However, as the load increases, the load balancing scheme starts to show a decrease in the number of bids since more rejections start to occur. Priority users submit more bid transactions than regular users, which submit many more search transactions, which require more resources to execute than bid transactions. In particular, at 1,500 clients, the PR scenario can process nearly 20% more bids than the LB scenario.

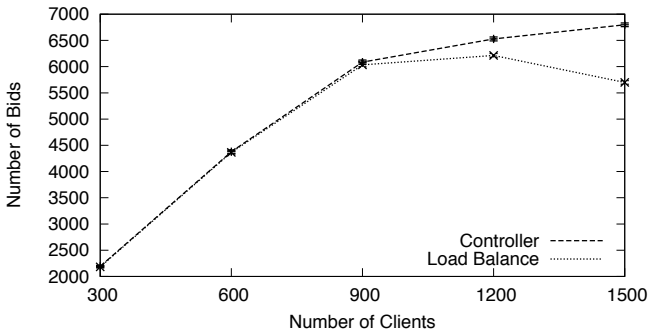

Figure 5. Number of bids for the LB and PR cases.

Figure 6 shows the variation of the response time versus the number of clients for both regular and priority users for both scenarios: LB and PR. The workload for priority and regular users is different. For example, regular users perform a large number of search transactions which usually take longer to run compared to other requests. Priority users go straight to their personalized *my auctions* type of page that shows all their active auctions; monitor the status of their auctions and place bids. These requests are typically faster than the browse and search transactions. For these reasons, when using a load balancer, the average response time of regular users is higher than that of priority users.

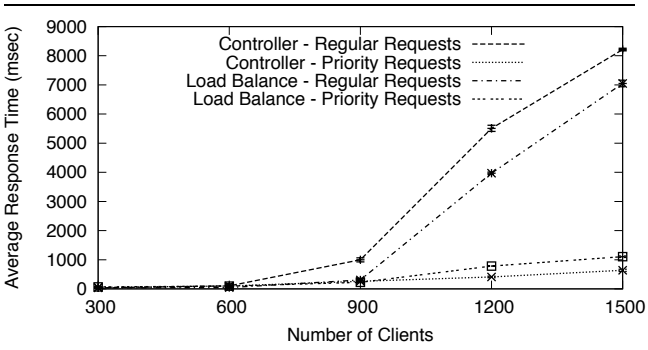

Figure 6. Response time for regular and priority requests for the PR and LB scenarios as a function of the load (in number of clients)

When using a dynamic load controller, the average response time of regular users is increased and that of priority users is reduced. The two inner lines in the graph of Fig. 6 are for the LB case and the two outer ones are for the PR case. Note that the lowest response time is obtained by priority users under the PR case. It is interesting to note the

actual values of the response time for the highest load (i.e., 1,500 clients): 0.63 sec for priority users under the PR case, 1.10 sec for priority users under the LB case (i.e., a 75% higher response time when compared with the PR case), 7.04 sec for regular users under the LB case, and 8.21 sec for regular user under the PR case. Note that the significant improvement in response time for priority users under the PR case comes at the expense of a small increase (i.e., 16.6%) in the response time of regular users.

It is interesting to examine the abandonment rate of regular and priority users. Figure 7 displays the percentage of rejected requests (i.e., requests that are rejected because they took longer than 5 seconds to respond) for various workload intensity levels and for both scenarios: LB and PR. As the load increases, the abandonment rate increases for both LB and PR but it is always higher for the PR case since regular users are given a lower priority under this scheme while they are treated equally as priority users in the LB scheme. It should also be noticed that at very high loads, the abandonment rate of regular users can get quite high, around 30%.

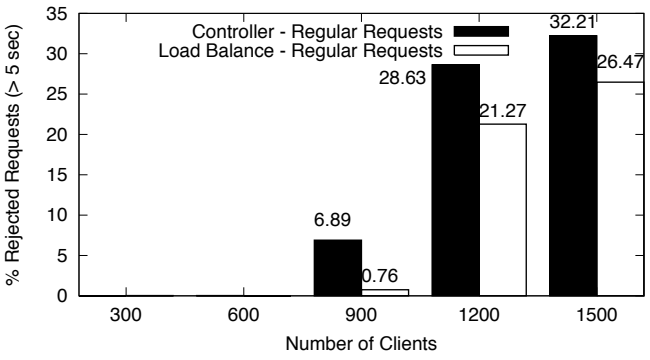

Figure 7. Percent of rejected request for regular users for the PR and LB cases as a function of the load (in number of clients)

The situation is quite different though for priority users as shown in Fig. 8. As it can be seen, priority requests always have a lower abandonment rate in the PR case when compared to the LB case. For example, at a very high load of 1,500 clients, the abandonment rate for priority users is 2.56% under the PR scenario and 4.17% for the LB scenario. That means that under the LB case, about 63% more priority user transactions are lost. If we consider that these customers are the ones who bid more often (see Table 2), the auction site could potentially lose significant revenue under the LB scenario.

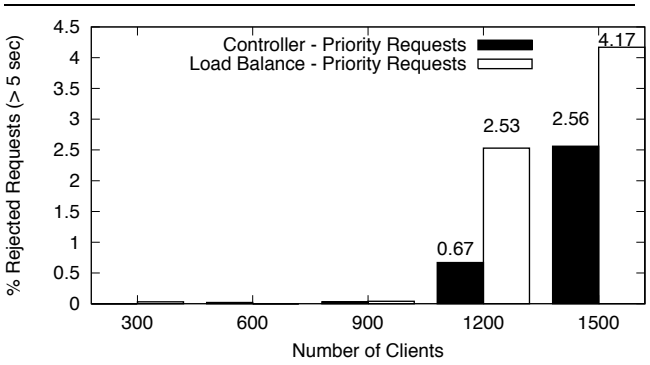

Figure 8. Percent of rejected request for priority users for the PR and LB cases as a function of the load (in number of clients)

Figure 9 depicts the utilization of the CPU for both application servers, priority and regular, under the PR and LB scenarios. Several observations can be made from that figure. First, as expected, the CPU utilization increases with the load but it increases at a smaller rate at a heavy load than at a light load because of request rejections. The utilization of both servers is, as expected, virtually the same under the LB case. However, the utilization of the regular server is higher under PR than that of the priority server under PR. For example, at very high loads (i.e., 1,500 clients), the utilization of the regular application server under PR is 87.6% while the utilization of the priority server under PR is 70.6%. Under LB, both servers have a utilization of 83% under high load. This explains why priority users get a significantly better response time as shown in Fig. 6.

Table 3 shows the percentage of time during an experiment in which the priority server and regular server where found in each of the utilization range combinations. The table shows that for most of the time, the priority server never operated at a utilization level above that of the regular server. Also, when the utilization of the regular server was Very High, the priority server's utilization was Very High or High in over 60% $((13.33 + 18.92)/52.43)$ of the cases because in these cases the priority server will not be taking much load from the regular server. It is also interesting to observe from the column and row totals that the priority server is in High or Very High utilization during 37.12% of the time while the regular server is in either state for 71.89% of the time.

It is worth comparing Tables 3 and 1. As it can be seen, the largest value in each column of Table 3 in almost all cases (except the Medium column) correspond to the "no change" cell in Table 1, which shows that the dynamic controller attempts and succeeds to bring the system to the de-

Priority Server Utilization ↓	Regular Server Utilization →					
	Very High	High	Medium	Low	Very Low	Total
Very High	13.33%	1.62%	0%	0%	0%	14.95%
High	18.92%	3.24%	0%	0%	0%	22.16%
Medium	12.79%	9.01%	1.08%	0%	0%	22.88%
Low	5.23%	1.98%	3.24%	2.52%	0%	12.97%
Very Low	2.16%	3.60%	5.95%	10.09%	5.23%	27.03%
Total	52.43%	19.46%	10.27%	12.61%	5.23%	100%

Table 3. Percent of time spent in each utilization configuration.

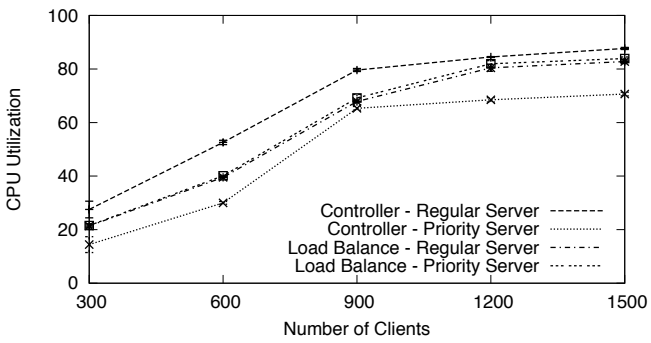

Figure 9. CPU utilization as a function of load (in number of clients) for the regular and priority server under the PR and LB scenarios

sired state.

One might argue that prioritizing only revenue generating transactions such as bids or auctions might lead to higher overall revenue regardless of which type of user generated these requests. We also evaluated service differentiation based on transaction type and found that these policies did not translate to significantly higher overall number of bids and auctions. Several non-revenue generating transactions including searching and viewing lead to revenue generating transactions such as auction creation and bidding. Lower response times with non-revenue generating transactions lead to losing customers because of slower response times during these activities. This eventually translates to lower overall revenue for the site. By prioritizing users based on revenue potential, all transactions originating from these users are processed faster, with minimum overall abandonment rate, leading to higher overall revenue generated by the auction site as shown in this paper.

7. Related Work

Menascé et al. demonstrated the advantages of using resource allocation techniques in e-commerce sites that are aimed at improving the revenue throughput (i.e., dollars generated per second) as opposed to managing resources based on performance criteria [18]. Their work used a simulated online bookstore to show the effect of policies that allocate CPU and I/O resources based on the content of user's shopping carts. The work presented in this paper is similar to that work in the sense that the criteria used for resource allocation are based on the type of user (i.e., users that bid more and therefore generate more revenue vs. users who bid less frequently). While we are inspired by that work, we differ in various important aspects: i) the resources we consider are the application servers of an auction site, ii) we consider very specific and realistic workloads typical of auction sites, and iii) our evaluation is based on an actual implementation of our ideas and not on simulation.

There has been significant work on load balancing for server clusters, mainly aimed at optimizing performance. It would not be possible to cite all of them here. However, we reference a few representative papers and we warn the reader that this is by no means an exhaustive list. Several techniques for improving web performance using dynamic load balancing techniques were discussed in [12, 8]. Schroeder and Harchol-Balter et al. [21] proposed kernel-level modifications to web servers to change scheduling policies to improve web server performance when the load fluctuates over time. A technique to improve performance of web sites that deliver dynamic content was proposed in [5] and is based on the notion of degrading dynamic content based on elapsed time of content generation. Similar content adaptation technique was presented in [1], in which the site responds to users with minimum information necessary by switching to basic mode without extensive personalization and graphic information. Several alternative architectures, load sharing policies, and dispatching request algorithms for designing and implementing scalable Web-server systems were discussed in [7].

Harchol-Balter et. al. present a task allocation mechanism for server clusters called cycle stealing under immediate dispath [10]. Similarly to our work, they assume the existence of two types of servers—a donor server and one or more beneficiary servers. They present an elegant analytical derivation of the response time of requests assuming Poisson arrivals. Requests are sent to one of these servers by a front-end dispatcher that uses the following rule: requests aimed at the donor server are sent to it but requests aimed at beneficiary servers are sent to a beneficiary server unless they are all busy and the donor server is idle. While apparently similar to our approach, there are some important differences: i) our approach does not make a dispatching decision on a request-by-request basis; the dispatching rule changes only when the value of f changes according to a much more elaborate decision matrix (see Table 1); ii) we view the workload as composed of sessions and not by isolated requests; iii) our work considers a 3-tier auction site as opposed to a single-tier web cluster; iv) we evaluate our policy using real experiments using a realistic workload as opposed to a Poisson arrival process; and v) our work is aimed at optimizing business-oriented metrics instead of just focusing on response time. A similar situation of scheduling two types of requests to two types of servers was considered by Ahn et. al. However, they looked at the problem from a point of view of a closed system with no arrivals and examine the question of when it is optimal for server 2 to aid server 1 with type-1 jobs rather than process type-2 jobs in order to minimize the total holding costs incurred until all jobs in the system are processed and leave the system [2]

Other approaches to resource management include either request based or session based admission control policies as indicated in [9, 11, 13]. For example, once the auction site is under heavy load conditions, instead of rejecting all the requests, it can selectively accept the revenue-generating requests mentioned above. Similar techniques which use priority scheduling for revenue generating transactions were proposed in [22, 23].

8. Concluding Remarks

This paper provided a *business-oriented* approach to addressing the problem of allocating servers to incoming requests of an auction site. Traditionally, load dispatching techniques are aimed at optimizing conventional performance metrics such as average response time without considering the nature of the requests and their business value. The approach considered in this paper is aimed at improving the revenue of an auction site by providing better performance to groups of users that have higher business value at the expense of other less important users.

More specifically, we presented the design of a dynamic load dispatcher for application servers that employs business-oriented criteria to route requests to the various application servers. Users are divided into two categories: priority (generate more revenue to the site than other users) and regular (generate less revenue than priority users). Application servers are also divided into priority and regular servers. Requests from priority users are treated exclusively by priority servers. Requests from regular users go primarily to regular servers, but as the load on the priority servers is reduced, some of the load of regular requests is shifted to the priority servers. The controller dynamically adjusts how much regular traffic should be diverted to priority servers. A variation to this approach could allow for load from priority users to be shifted to regular servers if the load at priority servers becomes excessive. A related issue worth investigating is how to determine the optimal value for the number of priority servers P given the total number of servers N. Autonomic computing techniques such as the ones described in [4] could be used to dynamically adjust the number P.

We implemented a testbed for auction sites that is an extension of RUBiS and uses realistic workload generators. Our dynamic load controller was implemented in the testbed and compared with a regular load balancer. We showed that the dynamic controller can increase the number of bids processed by the auction site by nearly 20% even at high loads. Lower abandonment rate of higher revenue generating customers along with higher number of total number of bids processed indicate that the revenue of auction sites can be improved using service differentiation based on user priorities. The priority of users can be identified based on various policies such as those based on the value of auctions, users historical auctions, and users buying patterns. Similarly, users can also be classified into more classes instead of priority and regular classes to provide differentiated services. Our load dispatching framework can be used to evaluate various policies to identify their effectiveness by extracting relevant metrics since we store the complete auction and bid information in the database.

References

[1] T.F. Abdelzaher and N. Bhatti, "Web content adaptation to improve server overload behavior," *Computer Networks*, Amsterdam, Netherlands, Vol. 31, No: 11–16, pp. 1563–1577, 1999.

[2] H. Ahn, I. Duenyas, and R.Q. Zhang, "Optimal control of a flexible server," *Adv. in Appl. Probab.*, Vol. 36, No. 1 (2004), pp. 139-170.

[3] V. Akula and D.A. Menascé, "Two-Level Workload Characterization of Online Auctions,"' *Electronic Commerce Research and Applications Journal*, Elsevier, in press.

[4] M.N. Bennani and D.A. Menascé, "Resource Allocation for Autonomic Data Centers Using Analytic Performance Mod-

els," *Proc. 2005 IEEE International Conference on Autonomic Computing*, Seattle, WA, June 13-16, 2005.

[5] L. Bradford, S. Milliner, and D. Dumas, "Scaling Dynamic Web Content Provision using Elapsed Time Based Content Degradation, *Proc. of Web Information Systems Engineering Conf.*, 2004.

[6] D. Bunnell, *The e-Bay Phenomenon: Business Secrets Behind the World's Hottest Internet Company*, Wiley, September 2000.

[7] V. Cardellini, E. Casalicchio, M. Colajanni, and P.S. Yu, "The state of the art in locally distributed Web-server systems," *ACM Comput. Surveys*, Vol. 34, No. 2, 2002, pp. 263–311.

[8] V. Cardellini, M. Colajanni, and Philip S. Yu, "Dynamic Load Balancing on Web-Server Systems," *IEEE Internet Computing*, Vol. 3, No. 3, pp. 28–39, 1999.

[9] L. Cherkasova, P. Phaal, "Session Based Admission Control: a Mechanism for Peak Load Management of Commercial Web Sites," IEEE J. Transactions on Computers, (TOC), Vol. 51, No. 6, June 2002.

[10] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M.S. Squillante, "Cycle Stealing under Immediate Dispatch Task Assignment," *Proc. Fifteenth ACM SPAA*, June 7-9, 2003, San Diego, CA, pp. 274-285.

[11] S. Elnikety, E. Nahum, J. Tracey and W. Zwaenepoel, "A method for transparent admission control and request scheduling in e-commerce web sites," Proc. 13th International Conference on World Wide Web, New York, NY, 2004.

[12] A. Iyengar, E. Nahum, A. Shaikh, and R. Tewari, "Enhancing Web Performance," *Proc. 2002 IFIP World Computer Congress (Communication Systems: State of the Art)*, 2002.

[13] A. Kamra, V. Misra and E.M. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered Web sites," Twelfth IEEE International Workshop on Quality of Service (IWQoS 2004), Montreal, Canada, June 2004.

[14] D.A. Menascé and V. Akula, "Improving the Performance of Online Auctions Through Server-side Activity-Based Caching," *World Wide Web Journal*, in press.

[15] D.A. Menascé and V. Akula, "Server-side caching strategies for online auction sites," *Proc. 6^{th} Intl. Conf. Web Information Systems Engineering (WISE'05)*, New York, NY, November 20-22, 2005.

[16] D.A. Menascé and V. Akula, "Improving the Performance of Online Auction Sites through Closing Time Rescheduling", *1st International Conference on Quantitative Evaluation of SysTems (QEST-2004)*, Enschede, the Netherlands, September 27-30, 2004.

[17] D. A. Menascé, V. Almeida, R. Fonseca, and M. Mendes, "A Methoodology for Workload Characterization for E-Commerce Servers," *Proc. 1999 ACM Conference in Electronic Commerce*, Denver, CO, Nov. 3-5, pp 119-128.

[18] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes, "Business-oriented Resource Management Policies for E-Commerce Servers," *Performance Evaluation*, Elsevier, Vol. 42, Nos. 3-4, Oct. 2000, pp. 223–239.

[19] D. A. Menascé and V. Almeida, *Scaling for E-Business: technologies, models, performance and capacity planning*, Prentice Hall, Upper Saddle River, NJ, 2000.

[20] D. A. Menascé, V. A. F. Almeida, R. Riedi, F. Pelegrinelli, R. Fonseca, and W. Meira Jr., "In Search of Invariants for E-Business Workloads," Proc. Second ACM Conference on Electronic Commerce, Minneapolis, MN, October 17-20, 2000.

[21] B. Schroeder and M. Harchol-Balter, "Web servers under overload: How scheduling can help," Technical Report CMU-CS-02-143, Carnegie-Mellon University, 2002.

[22] N. Singhmar, V. Mathur, V. Apte and D. Manjunath, "A Combined LIFO-Priority Scheme for Overload Control of E-commerce Web Servers," Proc. IEEE RTSS International Infrastructure Survivability Workshop, Lisbon, Portugal, December 2004.

[23] A. Totok and V. Karamcheti, "Improving Performance of Internet Services Through Reward-Driven Request Prioritization," Fourteenth IEEE International Workshop on Quality of Service (IWQoS 2006), New Haven, CT, June 19-21, 2006.