

## Improving the Performance of Online Auction Sites through Closing Time Rescheduling

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

*The workload of auction sites exhibits some interesting features as indicated in previous work by the authors. Of particular importance to this paper is the fact that over thirty percent of the bids of an auction arrive in the last five percent of the auction's lifetime. This creates a surge in the load seen by auction sites as an auction's closing time approaches. The site's performance is degraded if the site is not able to cope with these load spikes. With performance degradation comes frustrated customers and lost business. To mitigate this problem, we propose that auction sites reschedule auction closing times within a relatively small window centered around the closing time originally proposed by a customer. The scheduler's goal is to more evenly distribute the number of closings of auctions in order to produce a more uniform distribution of the number of bids on the auction site. This paper describes the auction closing time scheduling algorithm, applies it to a trace of auctions and bids obtained from an actual auction site, and uses a simulation model to study the effects of the rescheduled trace on the site's performance. In addition to that, the paper presents an algorithm to obtain a complete knowledge (CK) but non-realizable schedule. This schedule could be used as a benchmark for comparison with the realizable rescheduled load.*

### 1. Introduction

Many traditional auction businesses are moving into the online auctions space joining well-established contenders, such as eBay and Yahoo!Auctions [3, 11]. The workload of auction sites exhibits some interesting features as indicated in previous work by the authors. Of particular importance to this paper is the fact that over thirty percent of the bids of an auction arrive in the last five percent of the auctions lifetime. This creates a surge in the load seen by auction sites

as an auction's closing time approaches. The site's performance is degraded if the site is not able to cope with these load spikes. With performance degradation comes frustrated customers and lost business.

Building websites that can scale well is one of the challenges faced by architects of e-commerce sites. Benchmarks can be used to compare competing site architectures. TPC-W benchmark [10], a benchmark for e-commerce sites, mirrors the activities of e-tailers such as online bookstores. An e-commerce site developed for auctioning goods online, such as eBay, exhibits significantly different characteristics from e-tailers. Thus, TPC-W is not well-suited for auction-related e-commerce sites. A specific benchmark for auction sites, RUBiS, was developed at Rice University [1]. This benchmark is not as thorough as TPC-W and its workload generation process does not reflect the characteristics of a real auction site. The results of this paper can be applied to design auction sites that scale well and support the spikes typically seen in the workload of these sites. The mechanisms presented here can potentially increase the revenue throughput [8] (measured in dollars/sec) generated by auction sites.

A good understanding of the workload of auction sites provides insights about their activities and helps in the process of designing business-oriented metrics and designing novel resource management policies based on these metrics, as done in [8]. Our previous workload characterization [6] includes a detailed analysis of real data, obtained through automated agents, from online auctions to uncover patterns related to their major activities. We also analyzed how the features of the workload change within price clusters determined by some specific rules. That work also yielded several results regarding winner activity, unique bidders, price variation, and closing time activity. The results relevant to this paper are as follows.

- Most of the bidding activity of an auction happens at the later stages of its life time. In fact, over 30% of

the bids arrive towards the last 5% of the auction's life time.

- Most winners enter an auction towards the closing time, place relatively few bids (most of the times only one bid) to win the auction. When they win the auction with more than one bid, they usually start with less than half of the final price with good participation in the auction.
- Prices rise much faster after 80% of the auction's life time has elapsed than during the earlier stages of the auction.

This paper proposes that auction sites reschedule auction closing times within a relatively small time window centered around the closing time originally proposed by a customer. The scheduler's goal is to more evenly spread the number of closings of auctions in order to produce a more uniform distribution of the number of bids on the auction site. A closing time rescheduling algorithm is described and applied to a trace of auctions and bids obtained from an actual auction site. A simulation model is used to study the effects of the rescheduled trace on the site's performance. In addition to that, the paper presents a complete knowledge (CK) algorithm to obtain a non-realizable schedule. This schedule will be used in our future work as a benchmark for comparison with the realizable rescheduled load.

The rest of this paper is organized as follows. Section two defines some basic concepts and notation used throughout the paper. Section three describes the data collection process. The next section presents the closing time rescheduling algorithm. Section five presents an experimental evaluation of the algorithm. Section six discusses a non-realizable complete knowledge rescheduling algorithm. Finally, section seven presents some concluding remarks.

## 2. Basic Concepts

An online auction is a method of selling on the Internet in a public forum through open and competitive bidding. A bid is a prospective buyer's indication or offer of a price he or she will pay to purchase an item at an auction. A proxy bid consists in submitting a confidential maximum bid to a proxy agent run by the auction site on behalf of a prospective buyer. The proxy agent automatically increases the bid to maintain the highest bid every time the prospective buyer is outbid, as long as the maximum bid is not reached. Proxy bidding stops when the bid has won the auction or reached the limit of the proxy bid.

Figure 1 shows a time line divided into time slots of duration  $s$  seconds each. A time slot (one minute in our case) represents the time granularity of our log. The auction opens at time  $t_o$  and its creator requests that it be closed at time  $t_c$ .

The online auction site may suggest to the user a rescheduled closing time  $t_c^*$  in the interval  $[t_c - W/2, t_c + W/2]$ . This interval is called the *rescheduling window*. The life time of an auction is the difference  $t_c^* - t_o$  between the time,  $t_c^*$ , at which the auction is scheduled to close and its opening time,  $t_o$ .

The age,  $A$ , of an auction is defined as the percentage of time elapsed since the auction's opening time relative to its lifetime. In other words,  $A = (t_{\text{current}} - t_o) / (t_c^* - t_o)$ .

## 3. Data Collection Process

A much larger and complete data set, as compared to our previous work [6], was collected for the purposes of this paper. We present here a brief summary of the data collection process. We collected data for auctions created during the month of January 2003 from the Yahoo!auctions site using automated data collection agents (see Figure 2).

The data collection agent was designed based on the fact that for this particular online auction site, auction information is available online after an auction closes and can be fetched using a URL that is constant except for a sequential auction ID embedded in the middle of the URL. A Unix cron job invokes the data collection agent at programmed regular time intervals. The data collection agent submits HTTP requests to the auction site with a dynamically generated URL that contains the next auction ID to collect. The retrieved auction HTML pages are parsed to extract the auction and bid information. The output of the parsing program is used as input to the Oracle SQL Loader program, which loads the information into a relational database. The information collected for each auction includes its opening and closing times, price information, the list of all bids placed during the auction (including bidder id), price and time of each bid. Several SQL queries were written against the database to identify significant patterns and generate the data used in this paper.

The data collection agent gathered a total of 344,314 auction items belonging to over two thousand categories. A total of 1.12 million bids were placed on these auctions before their closing time, which varied from the same day of opening to 90 days from the opening date.

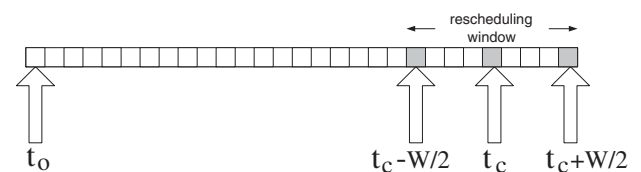


Figure 1. Auction closing and opening times.

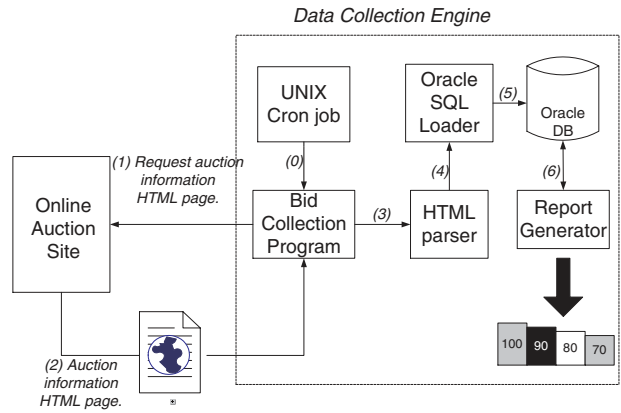


Figure 2. Data Collection Engine

Table 1 shows some summary statistics for the auctions monitored during the data collection period. Proxy agents placed 57% of all bids and bidders placed the remaining 43% bids manually. Forty one percent of the auctions received at least one bid and 39% of the auctions had a winner. Some auctions had a reserve price, which allows the seller to cancel the auction if no bids above the reserve price are placed. Auctions that receive some bids but do not have a winner can be attributed to cancellations due to reserve prices. For the experiments conducted in this paper, only auction creation requests and manual bid requests were considered. Proxy bid requests were not included because they are automatically generated by the auction site when manual requests are submitted. Since proxy bids are triggered by manual bids, we consider that the bid processing time includes processing time of all the proxy agent bids triggered by that manual bid.

4. A Closing Time Rescheduling Algorithm

The main rationale behind the closing time rescheduling algorithm is to attempt to more evenly distribute the bid load among the various time slots. Our previous work indicated that there is significant bidding activity in an auction as the closing time approaches. This is indicated in Fig. 3,

Total Auctions	344,314
Total Bids	1,125,183
Manual Bids	485,727 (43%)
Bids by Proxy Agents	639,456 (57%)
Auctions with at least 1 bid	140,039 (41%)
Auctions with a winner	133,121 (39%)

Table 1. Summary of Data Collected (January 2003)

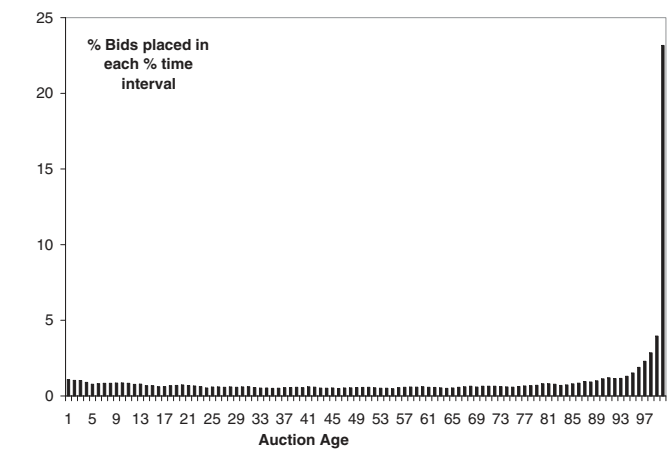


Figure 3. Closing time activity.

which shows the percentage of bids placed in each 1/100-th of an auction's life time for the extended data set collected for this paper. The figure shows that 30% of the bids arrive in the last 3% of the auction's life time.

As a first approximation, the closing time rescheduling algorithm may be described as selecting a new closing time for an auction so that it falls in the time slot with the smallest predicted load within the rescheduling window, of size  $W$ . The load at a given time slot is defined as the number of bid requests, from all open auctions, that fall in that time slot. To be more precise, the auction site keeps track of the predicted site load,  $SiteLoad(i)$ , for each future interval until the closing time of the auction to close further in the future. Then, the new closing time  $t_c^*$  is selected, as a function of the original closing time  $t_c$  and rescheduling window  $W$ ,

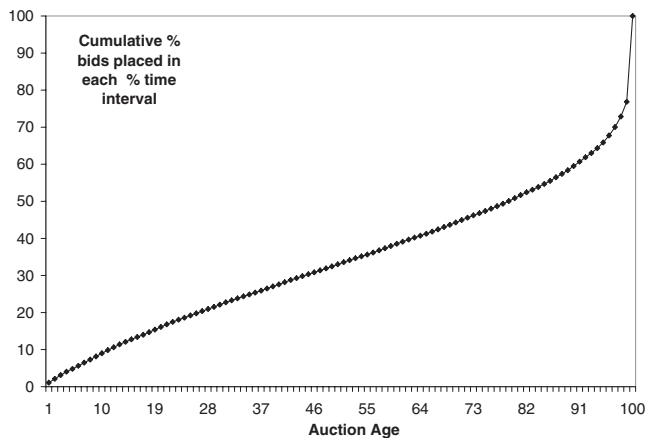
as

$$t_c^* = t \in [t_c - \frac{W}{2}, t_c + \frac{W}{2}] \text{ s.t.} \\ \text{SiteLoad}(t) = \min_{i \in [t_c - \frac{W}{2}, t_c + \frac{W}{2}]} \text{SiteLoad}(i). \quad (1)$$

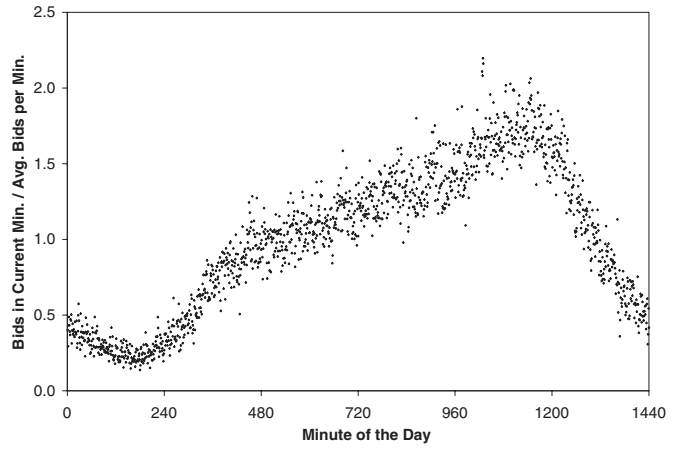
In reality, the rescheduling algorithm is a bit more involved as we explain below. The algorithm uses the results shown in Fig. 3 as an aid to predict the future bid load generated by an auction. This prediction is refined with every new bid as follows. Let  $f(a)$  be the expected percentage of bids received at age  $a$ . This function is the cumulative percentage derived from Fig. 3 and is shown in Fig. 4. Suppose that  $n$  bids have been received at age  $a$  of an auction. So, the total number of bids, TotalBids, for that auction is  $n \times 100 / f(a)$ . Thus, the projected number of bids, ProjectedBids, for any age  $a' > a$  given that  $n$  bids were submitted up to age  $a$  is

$$\text{ProjectedBids}(a', n, a) = n \times \frac{f(a')}{f(a)}. \quad (2)$$

Another consideration in the rescheduling decision is the time-of-day effect. As indicated in our previous work [6], there is a two-fold increase in the arrival rate of bid submissions towards the end of the day. This factor is taken into account when projecting the future site load. We define  $\text{TimeFactor}(i)$  as the ratio between the number of bids at time slot  $i$  and the average number of bids per minute. Figure 5 shows the value of  $\text{TimeFactor}$  for each minute of a day for the various days in our log. This factor is used to modify the predicted load during the last minutes of the auction. This way, the algorithm anticipates more bids towards the closing time if it falls in a peak time and less load otherwise.



**Figure 4. Function  $f(a)$ : percentage of bids received at age  $a$ .**



**Figure 5. Time of day factor.**

Our previous work showed that last minute bids are common and that in many auctions, only one or two bids are placed when the auction is about to close. This is called the “sniper effect.” Because of that, when an auction is created, one predicted bid is added to each minute of the last few minutes of the auction. This is called the *sniper load*. This way, even if no bids are submitted for the most part of the auction, the sniper load serves the purpose of avoiding too many auctions from being scheduled to close at the same time. As soon as a few bids (four in our case) are submitted, the sniper load is removed to avoid double counting the load at the last minutes of an auction. We used four bids as a threshold value because our data shows that this is the average number of bids submitted to auctions with at least one bid.

Figures 6 and 7 specify the algorithms used by an auction site to process a newly created auction and a newly submitted bid, respectively. When a new auction is created at time  $t_o$ , its original closing time  $t_c$  is rescheduled to a time  $t_c^*$  within a window of size  $W$  centered around  $t_c$  (see lines 2 and 3 of Fig. 6). Then, sniper load, is added to the 10 last closing time slots (i.e., the last 10 minutes) of an auction’s lifetime. The sniper load is equal to one unit of load adjusted by the time factor for each specific minute (see lines 6 and 7 of Fig. 6).

When a new bid arrives, the algorithm of Fig. 7 checks if there are enough bids to allow for the future bid load to be projected. The threshold is  $\text{MinBids}$ , which is equal to 4 in our experiments. If the current number of bids,  $\text{NumBids}$ , exceeds or is equal to the threshold, then the previously predicted load and the new projected load are computed for all time slots from the current time  $t$  until the closing time  $t_c^*$ . The site load is then updated by subtracting the previous load and adding the newly computed load. If the cur-

---

```

01 // Reschedule closing time
02  $t_c^* \leftarrow t \in [t_c - W/2, t_c + W/2]$  s.t. SiteLoad( $t$ ) =
03      $\min_{i \in [t_c - W/2, t_c + W/2]} \text{SiteLoad}(i)$ .
04 // Modify site load according to time-of-day factor.
05 For  $i = 1$  to 10 do
06     SiteLoad( $t_c^* - i$ )  $\leftarrow$  SiteLoad( $t_c^* - i$ )
07         + TimeFactor( $t_c^* - i$ )  $\times 1$ 

```

**Figure 6. New Auction Processing Algorithm**

---

rent number of bids is equal to MinBids, then the algorithm has to remove the sniper load that was added when the auction was created (see Fig. 6). The number of projected bids is computed with the help of Eq. 2 in lines 12 and 17 of Fig. 7. It should be noted that the rescheduling algorithm needs to keep track of the number of bids (NumBids) received so far on any auction and the age (AgeAtLastBid) of the auction when the last bid was received. The algorithm of Fig. 7 assumes that at age AgeAtLastBid, NumBids - 1 bids had been received. So, at age  $a$  (which corresponds to time  $t$ ) NumBids have been received.

The complexity of the bid processing algorithm is linear on the number of intervals from current time until auction's closing time. As indicated in Fig. 3, the number of bids in the early stages of the auction is much smaller than the number of bids in the closing minutes. This means, this algorithm iterates more times in the beginning stages of auctions for each of the few bids received. In the closing stages of the auction where the bidding activity is intensive, it has to iterate only over a few intervals.

## 5. Experimental Evaluation

In order to conduct an experimental evaluation of the auction time rescheduling algorithm presented in the previous section, we generate two traces from the data we collected. The first trace contains all auction creation requests and their bids in chronological order as submitted to the site during the month of January 2003. We call this the *original trace* in what follows. Then, we processed the original trace by applying the rescheduling algorithm described in the previous section to generate a *rescheduled trace*. When the closing time of an auction is rescheduled, the arrival times of the bids for that auction are changed in order to preserve the age at which they arrive. So, if a bid arrives at time  $t$  in the original trace, its arrival time,  $t'$ , in the rescheduled trace is

$$t' = t_o + \frac{(t_c^* - t_o)(t - t_o)}{t_c - t_o}. \quad (3)$$

---

```

01 // Compute the auction's age,  $a$ , at the current time  $t$ 
02  $a \leftarrow 100 \times (t - t_o) / (t_c^* - t_o)$ 
03 If NumBids  $\geq$  MinBids
04 Then For  $s = t + 1$  to  $t_c^*$  Do
05     Begin
06         // Compute the age  $a'$  at time slot  $s$ 
07          $a' \leftarrow 100 \times (s - t_o) / (t_c^* - t_o)$ 
08         // Compute the previous load at time slot  $s$ 
09         PrevLoad  $\leftarrow 0$ 
10         If NumBids  $>$  MinBids
11             Then // Previous load was predicted
12                 PrevLoad  $\leftarrow$  ProjectedBids( $a'$ ,
13                     NumBids - 1,
14                     AgeAtLastBid)
15             EndIf
16             // Compute the new load at time slot  $s$ 
17             NewLoad  $\leftarrow$  ProjectedBids( $a'$ , NumBids,  $a$ )
18             // Update site load
19             SiteLoad( $s$ )  $\leftarrow$  SiteLoad( $s$ ) - PrevLoad +
20                 NewLoad
21         End;
22 If NumBids = MinBids
23 Then // Remove sniper load
24     For  $i = 1$  to 10 Do
25         SiteLoad( $t_c^* - i$ )  $\leftarrow$  SiteLoad( $t_c^* - i$ ) -
26             TimeFactor( $t_c^* - i$ )  $\times 1$ 
27     EndIf
28 // Update age at last bid
29 AgeAtLastBid  $\leftarrow a$ 

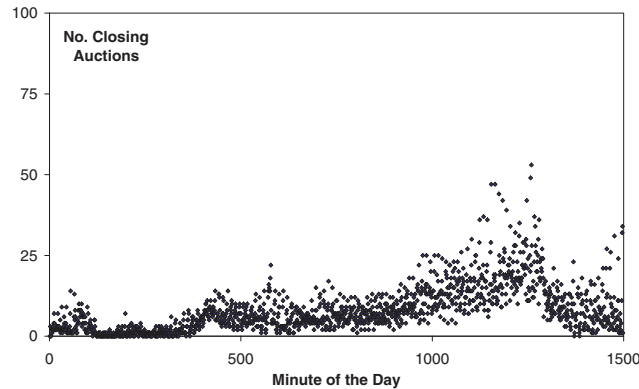
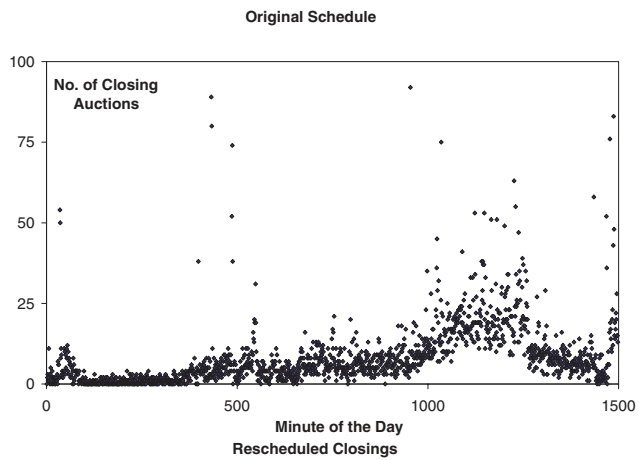
```

**Figure 7. New Bid Processing Algorithm**

---

Note that auction creation times ( $t_o$ ) are not changed by the algorithm. We used a scheduling window of one hour centered around the original closing time. This means the auction can close at most half an hour before or after the original closing time, which we believe would be acceptable to most of the users.

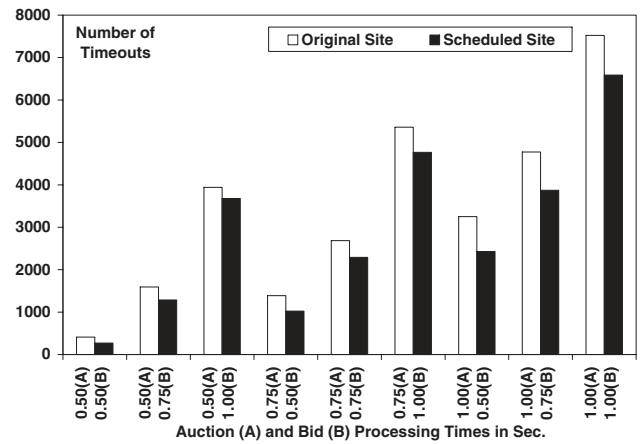
The first type of validation consisted in comparing the number of auctions closing at each interval for each of the two traces. Figure 8 shows the number of auctions closing in each minute of the day for the original trace (top graph) and for the rescheduled trace (bottom graph). The original trace exhibits several instances in which the number of closing auctions in a minute is much higher than in most minutes. For example, there are several minutes with 50 or more



**Figure 8. Number of auctions closing per minute in the original schedule (top) and in the rescheduled site.**

auctions closing on the same minute in the original schedule, while in the rescheduled case all minutes have less than 50 auctions closing.

The goal of the second type of evaluation was to assess the effect of a rescheduled set of auctions on the performance of an auction site. For that purpose, we built a trace-driven simulation of the auction site. The simulator, written in Java using simjava [4], reads a trace (the original or rescheduled trace) and simulates the processing of either an auction creation or a bid request. Requests for auction creation and bid processing join a queue. If a new bid request stays in the queue for more than a threshold equal to  $\tau$  seconds, the request is considered to timeout and is abandoned. Auction creation requests do not time out. Abandonment due to large response times is common in e-commerce environ-



**Figure 9. Number of bid requests that timeout for a timeout threshold of 3 seconds and an infinite queue.**

ments [5]. We simulated two situations: one in which the queue of requests is unlimited and the other in which there is a finite queue. In the finite queue case, arriving requests that find a full queue are rejected. The threshold  $\tau$  also applies to the limited queue case. In any case, the time to process an auction creation request was considered to be exponentially distributed with an average equal to  $\bar{x}_a$  seconds and the time to process a new bid was also considered to be exponentially distributed with an average equal to  $\bar{x}_b$  seconds. We considered the values of 0.5 sec, 0.75 sec, and 1.0 sec for  $\bar{x}_a$  and  $\bar{x}_b$ .

Figure 9 shows the number of bids requests that timeout for nine different combinations of the values of  $\bar{x}_a$  and  $\bar{x}_b$  and for the original and modified schedules. The value of  $\tau$  used in this figure is 3 seconds and the queue was considered to be unlimited. It should be noted that the threshold value represents the time spent at the auction site only and does not include network time. For that reason, we are using a value lower than the typical end-to-end abandonment thresholds considered elsewhere [5]. As the figure illustrates, the rescheduled site generates considerably less timeouts than the original schedule. For the more realistic cases where  $\bar{x}_a > \bar{x}_b$  the original schedule generates from 23% to 36% more timeouts than the rescheduled trace.

Table 2 shows the number of rejected requests for three combinations of  $\bar{x}_a$  and  $\bar{x}_b$  for a queue size limited at 20 requests. The timeout threshold  $\tau$  is also 3 seconds as before. The original trace rejects 5% to 8% more requests compared

$\bar{x}_a$	$\bar{x}_b$	Original:Scheduled (% Improvement)	
		Rejections	Timeouts
1	0.5	253:239 (5.53%)	3081:2284 (25.9%)
1	0.75	262:247 (5.72%)	4636:3658 (21.09%)
1	1	261:240 (8.04%)	7234:6359 (12.09%)

**Table 2. Number of bid requests rejected and timed out (queue size limit= 20; timeout threshold= 3 sec).**

to the rescheduled trace. In addition, the original trace generates from 12% to 25% more timeouts than the rescheduled trace for the limited queue size case. It should also be observed, as expected, that the number of timeouts is smaller for the limited queue case than for the infinite queue case. When the auction site is subject to the original trace, between six and nine percent more requests are rejected than when the rescheduled trace is used.

Unlike other e-commerce sites such as online book stores, if a request is rejected or timed out in the closing minutes of an online auction, it could mean permanently lost revenue. Users may not be able to go back and place their bid when the site is available as auctions are open only for a fixed amount of time. Both the rejected requests and the timed out requests translate to lost revenue to the auction sites. The results of these simulations indicate that online auction sites can improve revenue throughput by using rescheduling algorithms such as the one presented in this paper.

## 6. Complete-Knowledge (CK) Rescheduling

Given that we have a complete trace of auction creations and bid requests, one can think about deriving a non-realizable schedule that assumes the complete knowledge of when bids and auction creation requests will arrive in the future. Similarly to the OPT page replacement algorithm for operating systems [2], this CK schedule could be used as a benchmark to compare with realizable closing time rescheduling algorithms. We present in this section an algorithm to obtain a CK schedule.

Consider first the challenges of generating a CK schedule. When an auction is rescheduled, all of its bids need to be rescheduled so that they arrive at the original age.

Rescheduling the bids of auction  $a_i$  changes the future load and may impact other auctions that arrive after the opening time of auction  $a_i$  and before its maximum possible closing time.

However, if the maximum possible closing time of auction  $a_i$  (i.e., its original closing time +  $W/2$ ) is smaller than the opening time of the next auction, i.e., auction  $a_{i+1}$ , the rescheduling of auction  $a_i$  will not affect auction  $a_{i+1}$  nor any of its successors. One can then use this fact to break the entire original trace in non-overlapping strings. Each of these strings can then be rescheduled independently of one another.

We provide in what follows an algorithm to find these strings and the algorithm to reschedule each of these strings. Some notation is in order. Let

- $a_i$ : auction  $i$ .
- $n$ : number of auctions.
- $\text{closing}(a_i)$ : closing time of auction  $i$ .
- $\text{opening}(a_i)$ : opening time of auction  $i$ .
- $\mathcal{S}_j$ : the  $j$ -th string of overlapping auctions.

The algorithm to find all the strings  $\mathcal{S}_j$  is shown in Fig. 10. Each string  $\mathcal{S}$  generated by this algorithm can be processed independently of the others. Consider the following operations:

- $\text{first}(\mathcal{S})$ : returns the first auction inserted in the string  $\mathcal{S}$ .
- $\text{next}(\mathcal{S})$ : returns the next auction in the string  $\mathcal{S}$ . If there is no next auction, this operation returns  $\phi$ .

```

01  $j \leftarrow 1$ ;
02  $\mathcal{S}_j \leftarrow \phi$ ;
03 For  $i = 1$  to  $n$  do
04   begin
05      $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \{a_i\}$ ;
06     if  $i < n$ 
07       then if  $\text{closing}(a_i) + W/2 < \text{opening}(a_{i+1})$ 
08         then begin
09            $j \leftarrow j + 1$ ;
10            $\mathcal{S}_j \leftarrow \phi$ 
11         end
12   end

```

**Figure 10. Algorithm to find the interdependent strings of auctions**

The algorithm to reschedule all the auctions of a string  $S$  of interdependent auctions is shown in Fig. 11. The basic idea is to process a string from the first to the last auction. Each auction is then rescheduled as indicated in line 9 of Fig. 11. If no auctions get their closing times changed during a complete scan of the string, the algorithm completes. Otherwise, the string has to be rescanned. The algorithm in Fig. 11 includes a limit, MAXSCANS, on the number of scans. Stopping the algorithm before all possible scans may produce a lower quality schedule.

## 7. Concluding Remarks

This paper is the first, to our knowledge, to propose and evaluate a rescheduling algorithm of auction closing times in order to improve the performance of these sites. One important feature of the work reported here is that the motivation for the need to propose such mechanism arose from a detailed workload characterization of an actual auction site. Actual traces of this site were used to evaluate the efficacy of the proposed algorithm.

As part of future work we intend to derive, from the traces, different cumulative functions  $f(a)$  for the percentage of bids submitted at a given age  $a$  for several ranges of the total number of bids submitted to an auction. Then, as the number of bids of an auction increases, a different

---

```

01 Scanning ← true; NumScans ← 0;
02 While Scanning Do
03   Begin
04     a ← first( $\mathcal{S}$ );
05     Changed ← false; NumScans ← NumScans + 1;
06     While a  $\neq$   $\phi$  do
07       Begin
08         PreviousClosing ← closing (a);
09         reschedule (a);
10         If closing (a)  $\neq$  PreviousClosing
11           Then Changed ← true;
12         a ← next ( $\mathcal{S}$ )
13       End
14     If (Changed = false) or (NumScans > MAXSCANS)
15       Then Scanning ← false
16   End

```

**Figure 11. Algorithm to reschedule a string of interdependent auctions**

---

function  $f(a)$  will be used to provide better predictions on the future load. We intend to implement the CK rescheduling algorithm and use it to compare it with our realizable rescheduling algorithm.

## References

- [1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *Proc. IEEE 5th Annual Workshop on Workload Characterization (WWC-5)*, Austin, TX, November 2002.
- [2] E. Coffman and P. Denning. *Operating Systems Theory*. Prentice Hall, Upper Saddle River, NJ, 1973.
- [3] eBay. [www.ebay.com](http://www.ebay.com).
- [4] F. Howell and R. McNab. Simjava: A discrete event simulation package for java with applications in computer system modeling. In *Proc. First International Conference on Web-based Modeling and Simulation, Society for Computer Simulation*, San Diego, CA, January 1998.
- [5] D. Menascé and V. Almeida. *Scaling for E-Business: technologies, models, performance, and capacity planning*. Prentice Hall, Upper Saddle River, NJ, 2000.
- [6] D. A. Menascé and V. Akula. Towards workload characterization of auction sites. In *Proc. IEEE 6th Annual Workshop on Workload Characterization (WWC-6)*, Austin, TX, October 2003.
- [7] D. A. Menascé, V. Almeida, R. Fonseca, and M. Mendes. A methodology for workload characterization for e-commerce servers. In *Proc. ACM Conference in Electronic Commerce*, pages 119–128, Denver, CO, November 1999.

- [8] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. Mendes. Business-oriented resource management policies for e-commerce servers. *Performance Evaluation*, 42(223-239), September 2000.
- [9] D. A. Menascé, V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. M. Jr. In search of invariants for e-business workloads. In *Proc. Second ACM Conference on Electronic Commerce*, Minneapolis, MN, October 2000.
- [10] TPC. The tpc-w benchmark. [www.tpc.org](http://www.tpc.org).
- [11] Yahoo. Yahoo! auctions. <http://auctions.yahoo.com>.