Poster Session:
ASETS: A Self-Managing Transaction Scheduler *

Mohamed A. Sharaf[1] Shenoda Guirguis[2] Alexandros Labrinidis[2] Kirk Pruhs[2] Panos K. Chrysanthis[2]
[1] ECE Department, University of Toronto
[2] CS Department, University of Pittsburgh
msharaf@eecg.toronto.edu {shenoda, labrinid, kirk, panos}@cs.pitt.edu

Abstract

User satisfaction determines the success of web-database applications. User satisfaction can be expressed in terms of expected response time or expected delay. Given the bursty and unpredictable behavior of web user populations, we model user requests as transactions with soft-deadlines. For such a model of user requests with soft-deadlines, the hit ratio is not the most expressive metric. Instead, the average tardiness is a better measure in such cases. In this paper; we propose and evaluate an adaptive self-managing algorithm called ASETS*, which optimizes for the average tardiness.* ASETS *prioritize resources as needed in order to keep users satisfied under varying workloads. Our performance evaluation shows* ASETS *to outperform both* EDF *and* SRPT *which are known to be optimal for the under and over utilization system conditions respectively.*

1. Introduction

Web-database systems support nowadays the most prevailing e-services ranging from e-banking and e-commerce applications, to personalized news and weather services. In such highly interactive applications, user satisfaction or positive experience determines their success. Given the bursty and unpredictable behavior of web user populations, it is therefore crucial for such systems to be self-manageable, prioritizing resources as needed in order to keep users satisfied under varying workloads.

One way to model user experience (and therefore quantify user satisfaction) is to define for each requested service its "ideal" deadline to produce a result. After the deadline, the impact of additional delay (or *tardiness*) has a negative impact on user experience and needs to be minimized. This is an example of a Real-time Database System with *Soft Deadlines*, where "tardy" transactions are not dropped, and are allowed to complete their execution past their deadline. In such a system, the overall amount of tardiness is indicative of the overall user satisfaction.

Prioritizing or *scheduling* transactions in Real-time Systems in general, and Real-Time Database Management Systems (RTDBMS) such as Web-databases in particular, where the transactions have real-time constraints, has earned a lot of attention in the literature [1, 3, 7, 6, 10, 2, 4, 13, 9]. Of these schemes, the Earliest-Deadline-First (*EDF*) policy has been shown to be the best scheduling policy for light workloads (i.e., workloads that do not overwhelm the server). On the other hand, it has been shown that the Short-Remaining-Processing-Time (*SRPT*) policy is optimal for scheduling the processing of requests to Web servers [11], since it can handle overload situations much better. However, *SRPT* is oblivious to deadlines, so it would perform fare worse than *EDF* for light workloads.

An obvious integration of these two schemes assumes that one knows the "cross-over" point where *SRPT* is better than *EDF*. Such a point is not easy to determine (since this should be dependent both on the workload and the deadlines), and is also expected to change drastically over time, as the workload changes. In this paper, we propose *ASETS*, an adaptive policy for scheduling (query) transactions in the presence of soft deadlines, which does not assume any prior knowledge of the workload or the deadlines.

ASETS is a hybrid between *EDF* and *SRPT* that intelligently and dynamically splits the set of transactions currently in the system into those that should be scheduled using *EDF* and those to be scheduled using *SRPT*. *ASETS* actually stands for Adaptive *SRPT EDF* Transaction Scheduling. To the best of our knowledge, this is the first hybrid approach to minimize *average tardiness*, which does not require any tuning or parameters. Our experimental results using simulation show that *ASETS* outperforms both *SRPT* and *EDF* by up to 30% in average tardiness.

Road-map: The rest of the paper is organized as follows: Section 2 defines the system model. *ASETS* is motivated and explained in Section 3, and evaluated in Section 4. Section 5 discusses related work. We conclude in Section 6.

*This work was partially supported by NSF IIS-0534531.

2. System Model

We assume that our web database is an RTDBMS where each request is a transaction (T_i) associated with the following three characteristics:

- *Arrival Time* (a_i): The time when T_i has arrived at the RTDBMS;
- *Deadline* (d_i): The ideal time by which T_i should finish execution; and
- *Length* (l_i): The processing time needed to execute T_i.

The first two parameters above (i.e., a_i and d_i) are made available to the RTDBMS once a transaction T_i is submitted to the system, whereas the third parameter (i.e., l_i) is computed by the RTDBMS based on previous statistics and profiles of transaction execution.

The time when transaction T_i finishes execution is denoted as *finish time* (f_i). Ideally, f_i should be equal to the sum of a_i and l_i. However, this will only happen if transaction T_i does not experience any queuing delays or if it is the only transaction in the system, which is not the norm; a transaction will typically wait for other transactions to finish execution first, especially when the system is under high load.

For soft-deadline transactions, the RTDBMS strives to finish executing each transaction T_i before its deadline. However, if T_i cannot meet its deadline, the RTDBMS will still execute it but the system will be "penalized" for the delay beyond the deadline d_i. One natural transaction penalty is called *tardiness*, and one natural system performance metric is *average tardiness*. These are formally defined as follows:

Definition 1 *Transaction tardiness, t_i, for transaction T_i is the total amount of time spent by T_i in the RTDBMS beyond its deadline d_i. That is, $t_i = 0$ iff $f_i \leq d_i$, and $t_i = f_i - d_i$ otherwise. The average tardiness for N transactions is: $\frac{1}{N}\sum_{i=1}^{N} t_i$.*

RTDBMSs typically employ a transaction scheduler which decides the execution order of transactions. In order for a RTDBMS to perform well when transactions can have widely varying lengths, it is necessary that transactions are preemptable, that is, the scheduler can stop a certain transaction and switch to another transaction then resume the first transaction at a later time. One common and natural class of scheduling policies are called priority based policies. In a priority based policy, some priority p_i assigned to each transaction T_i, and the highest priority transaction is always executed first. Different schedulers consider different parameters for computing the priority p_i. Parameters to consider (beside those already mentioned above) include:

1. Remaining Processing Time (r_i): The current amount of time needed to finish processing T_i.
2. Slack Interval (s_i): The interval of time between T_i's expected finish time f_i and its deadline d_i if T_i is executed right now.

For example, under the *EDF* scheduling policy, each transaction is assigned a priority $p_i = 1/d_i$, while under *SRPT*, $p_i = 1/r_i$. In the next section, we will further explain the *EDF* and the *SRPT* policies, as well as our proposed *ASETS* scheduling policy.

3. Scheduling Policies

In this section, we first illustrate the trade-off between the two policies widely used for scheduling soft-deadline transaction, namely, *EDF* and *SRPT* in section 3.1. Then, in section 3.2, we introduce our proposed hybrid policy *ASETS* that better optimizes for user satisfaction, that is, average tardiness.

3.1. *EDF* vs *SRPT* in RTDBMS

With respect to average tardiness, two natural policies are Earliest Deadline First (*EDF*), and Shortest Remaining Processing Time (*SRPT*). These policies can be viewed as extreme policies on a spectrum of possible priority policies.

Under *EDF*, a transaction with an early deadline receives a higher priority, whereas under *SRPT*, a transaction with a shorter processing time is the one which receives higher priority. *EDF* guarantees that all jobs will meet their deadlines if the system is not over-utilized. As such, the tardiness of the system is expected to be zero since all the transactions make their deadlines. When the system is over-utilized, it is impossible to finish all transactions by the specified deadlines. So some transactions will experience tardiness. Using an *EDF* scheduler in such high-load situations will have a substantial negative impact on the overall tardiness. This negative impact is known as the *domino effect* where transactions keep missing their deadlines in a cascaded fashion. The cause of the domino effect is that *EDF* might give high priority to a transaction with an early deadline that it has already missed, instead of scheduling another one which has a later deadline that could still be met. As a result, both transactions will miss their deadlines and accumulate tardiness.

In contrast to *EDF*, *SRPT* is the best policy to use when all transactions have already missed their deadlines. This is because the problem of minimizing tardiness in this case is the same as the problem of minimizing response time, for which *SRPT* has been shown to be the optimal policy [11]. However, in the cases when there are transactions that have

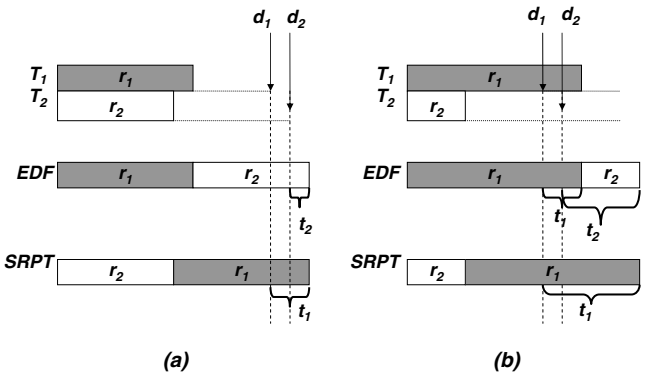

Figure 1. *EDF* **vs** *SRPT* **scheduling: (a) A case when** *EDF* **outperforms** *SRPT*, **(b) A case when** *SRPT* **outperforms** *EDF*

not missed their deadline yet, *SRPT* might run into the problem of assigning a high priority to a short transaction that has a long deadline instead of scheduling another transaction which is relatively longer out its deadline is imminent deadline.

Example 1: To further illustrate the difference between the two policies, consider the example illustrated in Figure 1. The figure shows two sets of transactions T_1 and T_2 with deadlines d_1 and d_2, and remaining processing r_1 and r_2, respectively.

In Figure 1(a), the tardiness of *EDF* ($= t_2$) is less than that of *SRPT* ($= t_1$). The reason for *SRPT*'s higher tardiness is giving higher priority to T_2 which has the shorter remaining processing time (r_2) but a longer deadline (d_2). For the other set of transactions in Figure 1(b), *EDF* provides higher tardiness ($= t_1 + t_2$). This is because it scheduled T_1 first which is already past its deadline leading to missing T_2's deadline as well.

It is known that there is no policy that can guarantee that the average tardiness is within any constant factor of the retrospective optimal average tardiness (the average tardiness that would have been attained if future transactions were known and every scheduling decision was correct/optimal) [5]. In fact, even if a policy could somehow magically process transactions 100 times faster, it still cannot guarantee average tardiness within a constant factor of optimal [5]. Further, even if one knew all the future transactions, computing the optimal average tardiness schedule is NP-hard even if all jobs have the same length [12].

3.2. The ASETS Scheduling Policy

As it is obvious from the previous example, there is no clear best policy for scheduling transactions with deadlines that minimizes tardiness under all workloads. Generally

Input: A set of transactions
Output: The id of the transaction to run till next scheduling point
BEGIN
Add all newly arrived jobs to *EDF-List*
for all $T_i \in$***EDF-List*** **do**
  if $(t + r_i > d_i)$ **then then**
    move T_i **to *SRPT-List***
  end if
end for
$T_{1,EDF} \leftarrow Top($***EDF-List***$)$
$T_{1,SRPT} \leftarrow Top($***SRPT-List***$)$
if $(r_{1,edf} < r_{1,srpt} - s_{1,EDF})$ **then**
  return id of $T_{1,EDF}$
else
  return id of $T_{1,SRPT}$
end if
END

Figure 2. The *ASETS* **Algorithm**

speaking, *EDF* performs well at low utilization, whereas at high utilization, *SRPT* performs better than *EDF*.

One possibility is to select the policy dynamically based on the load of the system. However, measuring the load with reasonable accuracy may require non-trivial resources. More importantly, when jobs have deadlines, measuring the load does not only involve considering the processing requirements of the transactions, but also the relationships between processing times and deadlines. For example, a batch of transactions with very low processing requirements but very tight deadlines will lead to an overloaded condition. Depending on how one formally defines load, the problem of determining the load of a collection of transactions with deadlines either requires solving a network flow or matching problem, or is NP-hard.

In this paper, we propose a hybrid policy for transaction scheduling called *Adaptive SRPT EDF Transaction Scheduling (ASETS)*. *ASETS* is a parameter-free adaptive policy which integrates the advantages of both the *SRPT* and *EDF* policies and automatically adapts to system load.

Under ASETS, the scheduler maintains two priority lists. In the first list, called *EDF-List*, transactions are ordered according to their deadlines as in the *EDF* scheduling policy. In the second list, called *SRPT-List*, transactions are ordered according to their remaining processing time as in the *SRPT* scheduling policy.

The first list, *EDF-List*, contains all transactions that can still make their deadlines. Formally,

Definition 2 *A transaction T_i with deadline d_i is included in* EDF-List *if and only if, $t + r_i \leq d_i$, where t is the current time.*

The second list, *SRPT-List*, contains all transactions that already missed their deadlines. Formally,

Definition 3 *A transaction T_i with deadline d_i is included in* SRPT-List *if and only if, $t + r_i > d_i$, where t is defined as above.*

Notice that each transaction starts in the *EDF-List* then it might move to the *SRPT-List* if it misses its deadline. Given the above two lists, at each scheduling point *ASETS* selects for execution either the transaction at the top of *EDF-List* or the one at the top of *SRPT-List*. For convenience, we will call these two transactions: $T_{1,EDF}$ and $T_{1,SRPT}$, respectively.

To decide between $T_{1,EDF}$ and $T_{1,SRPT}$, we use a simple greedy heuristic under which $T_{1,EDF}$ is scheduled for execution if $r_{1,EDF} < r_{1,SRPT} - s_{1,EDF}$, otherwise, $T_{1,SRPT}$ is the one scheduled for execution. The premise underlying this heuristic is to schedule the transaction with the least "negative" impact. In particular, if $T_{1,EDF}$ is scheduled first, its negative impact is increasing $T_{1,SRPT}$'s tardiness by $r_{1,EDF}$. On the other hand, if $T_{1,SRPT}$ is scheduled first, its negative impact is increasing $T_{1,EDF}$'s tardiness by $r_{1,SRPT}$ minus the amount of slack $s_{1,EDF}$ that $T_{1,EDF}$ currently has.

Example 2: Given the example previously illustrated in Figure 1, *ASETS* will produce a schedule similar to *EDF* for the transactions in Figure 1(a) since they are both in the *EDF-List*. For the transactions in Figure 1(b), *ASETS* will place T_1 in the *SRPT-List* and T_2 in *EDF-List* and it will schedule T_2 first since it is the one with minimum negative impact. This will result in the same scheduler produced by *SRPT* which minimizes the tardiness for that setting.

Finally, notice that given the above two list arrangements, in the extreme case where all transactions are past their deadlines, *ASETS* is basically equivalent to *SRPT*. In the other extreme case where all transactions can meet their deadlines, then *ASETS* behaves like *EDF*. In the general case, where there is a mix of transactions that have passed their deadlines and others that can still meet their deadlines, *ASETS* policy employs both *SRPT* and *EDF*. This allows our proposed hybrid policy, *ASETS*, to outperform *SRPT* and *EDF* as it is experimentally shown in the next section.

The policy *ASETS* need only be invoked in response to two types of events, the arrival of a job, and the completion of a job. Using a standard balanced binary search tree, only time $O(\log N)$ is required in each case.

4. Performance Evaluation

We have conducted multiple experiments to evaluate the performance of our proposed *ASETS* policy. We describe the settings for these experiments in section 4.1 and the experimental results in section 4.2.

Parameter	Meaning	Value
l_i	transaction length	Zipf(α) over [1 - 50]
α	skewness of job length distribution	0.5
k	slack factor	[0.0 - k_{max}]
a_i	arrival time	Poisson process with arrival rate = $\frac{SystemUtilization}{AvgTransactionLength}$
SystemUtilization		[0.1 - 1.0]

Table 1. Experiments parameters summary

4.1. Experimental Setup

We created an RTDBMS simulator and conducted several experiments to compare the performance of our proposed *ASETS* policy against the previously described *EDF* and *SRPT* policies. For completeness, we have also compared it against the traditional *First Come First Served (FCFS)* as well as the *Least Slack (LS)* policy [1], where under *LS*, the priority of transaction T_i is set to $1/s_i$.

We created a transaction workload similar to those in [6, 1]. Specifically, we generated 1000 transaction where the transaction length l_i is generated according to a Zipf distribution over the range [1–50] time units with the default Zipf parameter for skewness (α) set to 0.5 and it is skewed toward short transactions.

Each transaction is assigned a deadline $d_i = a_i + l_i + k_i \times l_i$ where k_i is a factor that determines the ratio between the initial slack time of a transaction and its length. k_i is generated uniformly over the range [0.0–k_{max}], where k_{max} is a simulation parameter with default value of 3.0.

The generated transactions arrive at the RTDBMS according to a Poisson distribution with arrival rate equal to $SystemUtilization/AvgTransactionLength$, where $SystemUtilization$ is a simulation parameter that takes the values [0.1–1.0].

The values of average tardiness reported in the next section are the averages of 5 runs for each experiment setting. We have conducted multiple experiments to examine all possible parameters' values, that are summarized in Table 4.1. In all our experiments, *ASETS* significantly outperforms the other scheduling policies. Below, we present a representative sample of our results.

4.2. Results

In our first experiment, we measured the average tardiness for the five scheduling policies mentioned above as the

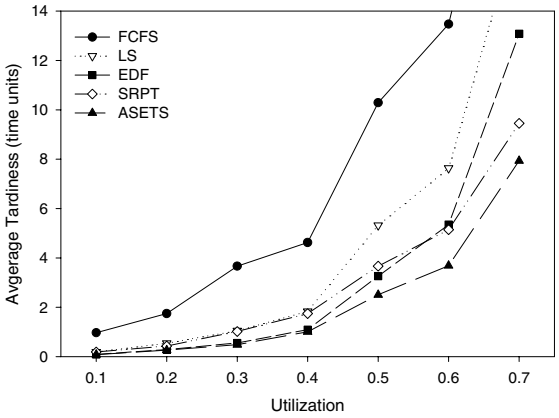

Figure 3. Avg Tardiness under Low System Utilization ($\alpha = 0.5$)

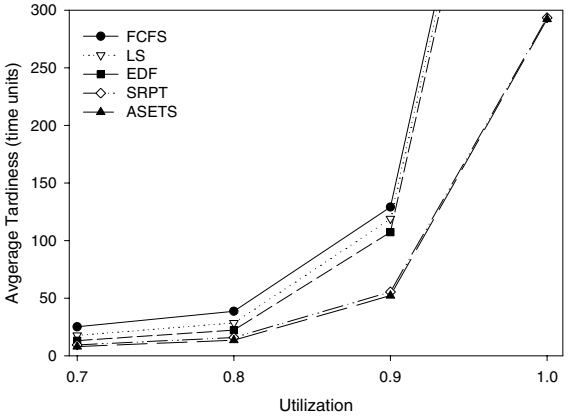

Figure 4. Avg Tardiness under High System Utilization ($\alpha = 0.5$)

system utilization increases from 0.1 to 1.0, with Zipf's parameter $\alpha = 0.5$ and $k_{max} = 3.0$.

The results for that experiment setting are shown in figures 3 and 4, where Figure 3 shows the average tardiness at low utilization while Figure 4 shows the average tardiness at high utilization; we split the utilization across two figures to zoom in for better understanding of the system behavior. Specifically, at low utilization (Figure 3), the system is able to meet most of the deadlines, and hence, *EDF* performs better than *SRPT*. As the utilization grows, the system cannot meet as much deadlines, and *SRPT* starts to approach *EDF* till it outperforms it at utilization 0.6.

ASETS on the other hand, outperforms both *EDF* and *SRPT* for all values of utilization. However, it is noticed that the maximum improvements provided by *ASETS* is around the cross over point between *EDF* and *SRPT* where it reduces the average tardiness by up to 30%. This is further illustrated in Figure 5, where we plot the average tardiness of *ASETS* normalized to that of *EDF* as well as *SRPT*.

Figure 5 also shows that *ASETS* outperforms *EDF* even at very low utilization values. The reason is that though the overall average utilization is low, there are still intervals where the utilization increases significantly above the average due to the fact that we are using Poisson arrivals. At those high utilization intervals, *ASETS* automatically incorporates some *SRPT* scheduling to avoid the domino effect of *EDF*. Similarly, at high utilizations, *ASETS* outperforms *SRPT* as it incorporates some *EDF* scheduling as needed.

Next set of results shows the performance of our proposed algorithm under different deadline settings. Specifically, we compare the performance of *ASETS* to *SRPT* and *EDF* under different values of k_{max}. Figures 6, 7, and 8 show the results for k_{max} values of 1, 2, and 4 respectively. This is in addition to the results of $k_{max} = 3$ presented above in Figure 5.

The results show that *ASETS* constantly outperforms the other two algorithms under the different settings, with the maximum gain be at the cross-over area. It is also interesting to notice that the cross-over point moves further to the right (i.e., higher utilization) as we increase the value of k_{max}.The reason is that the more loose the deadlines are (larger k_{max}) the more chances *EDF* can catch up if it missed deadlines. Hence *EDF* can cope with higher utilization and outperforms *SRPT* for a longer range of utilization.

5. Related Work

Previous research efforts have proposed several hybrid approaches for scheduling real-time transactions (e.g., [3, 6, 7, 4]). However, these approaches have mainly focused on maximizing the hit-ratio (i.e., the number of transactions that meet their deadlines) or maximizing the system gain when each transaction is associated with a *value* or *award*. Below, we discuss some of these hybrid approaches since they share some features with our proposed *ASETS* policy.

For instance, the work in [3] studies the performance of algorithms that use deadline only, value only, or a mix of both in assigning transaction priorities. Specifically, it studies the *Highest Value First (HVF)* and the *Highest Density First (HDF)* policies. It also proposes a hybrid policy called *MIX* which uses a linear combination between the value and the absolute deadline in order to maximize the hit-ratio.

Also, towards maximizing the hit-ratio, the work in [6] proposes a hybrid algorithm to schedule transactions in RT-DBMS. The algorithm divides transactions into two sets, one to be scheduled using *EDF*, and another to be scheduled randomly where the size of each list is determined based on feedback of the achieved hit-ratio. The work in [6] further extends the proposed approach to maximize sys-

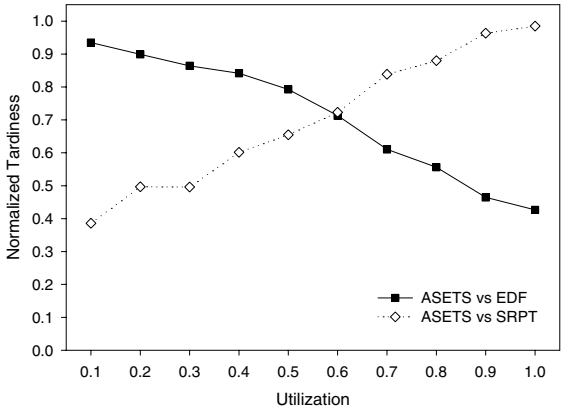

Figure 5. Normalized Average Tardiness ($k_{max} = 3$)

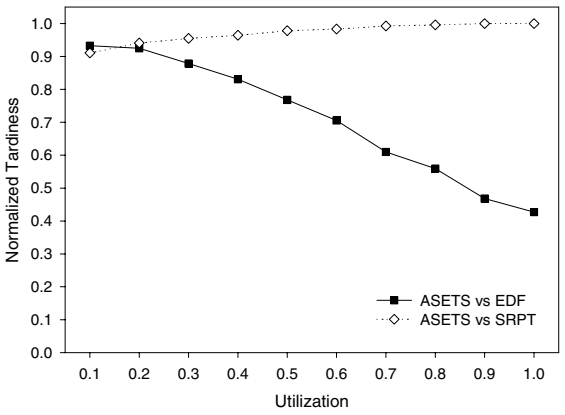

Figure 6. Normalized Average Tardiness ($k_{max} = 1$)

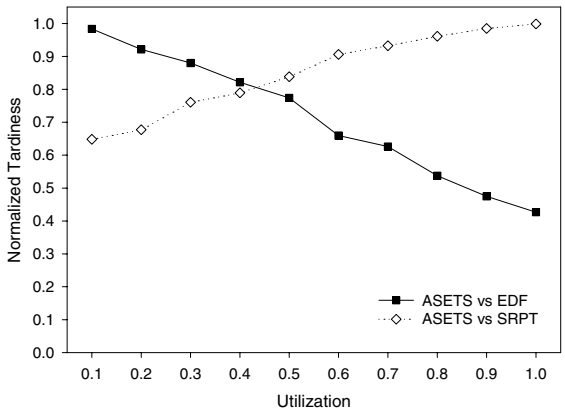

Figure 7. Normalized Average Tardiness ($k_{max} = 2$)

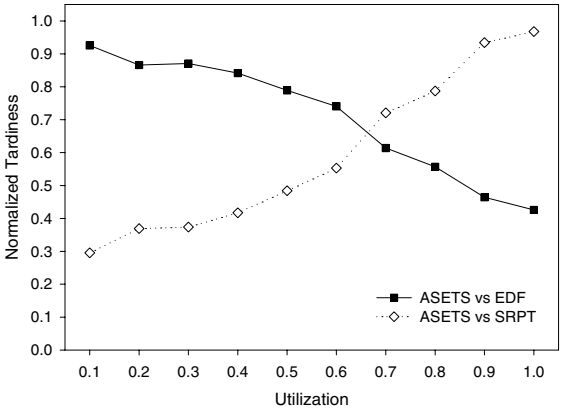

Figure 8. Normalized Average Tardiness ($k_{max} = 4$)

tem gain when transactions are associated with values.

In [4], another hybrid approach is proposed to schedule web-broadcasts. Basically, [4] proposes *MIA* which is a hybrid approach between *SRPT* and *EDF* that also considers the popularity of broadcast items to maximize the total system gain.

Previous work have also studied the interaction between transaction scheduling and concurrency control as in [10], as well as the trade-off between QoS and QoD as in [8].

6. Conclusions

Motivated by the need of an adaptive self-managing web-database system; in this paper we modeled user requests as transactions with soft-deadlines, and proposed a new scheduling algorithm called *ASETS*.

ASETS is based on *EDF* and *SRPT*. It intelligently combines the advantages of both *EDF* and *SRPT* scheduling algorithms under low and high system utilization conditions respectively. We evaluated ASETS experimentally and showed that our proposed approach outperforms both *EDF* and *SRPT* minimizing the average tardiness.

Possible future extension to this work is to examine how the algorithm will perform under the hard-deadline model. It would also be very interesting to generalize the model to consider update transactions.

In conclusion, it should be noted that *ASETS* is not limited to web-databases, but it can be applied in any Real Time system with soft-deadlines where minimizing tardiness is the right metric.

References

[1] R. K. Abbott and H. Garcia-Molina. Scheduling real-time transactions: a performance evaluation. *ACM TODS*, 1992.

[2] S. A. Aldarmi and A. Burns. Dynamic value-density for scheduling real-time systems. *ecrts*, 1999.

[3] G. Buttazzo, M. Spuri, and F. Sensini. Value vs. deadline scheduling in overload conditions. In *RTSS '95*.

[4] W. Cao and D. Aksoy. Beat the clock: a multiple attribute approach for scheduling data broadcast. In *MobiDE '05*, 2005.

[5] M. Chrobak et al. Preemptive scheduling in overloaded systems. In *Proceedings of the 29^{th} International Colloquium on Automata, Languages, and Programming*, 2002.

[6] J. R. Haritsa, M. Livny, and M. J. Carey. Earliest deadline scheduling for real-time database systems. In *Proceedings of RTSS '91*.

[7] D.-Z. He, F.-Y. Wang, W. Li, and X.-W. Zhang. Hybrid earliest deadline first /preemption threshold scheduling for real-time systems. In *ICMLC '04*.

[8] K.-D. Kang, S. H. Son, and J. A. Stankovic. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE TKDE*, 16(10), 2004.

[9] K.-Y. Lam, Y. Guo, M. Xiong, and B. Liang. Quality of service guarantee for temporal consistency of real-time transactions. *IEEE TKDE*, 18(8), 2006.

[10] Özgúr Ulusoy and G. G. Belford. Real-time transaction scheduling in database systems. *Inf. Syst.*, 1993.

[11] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Trans. Inter. Tech.*, 2006.

[12] Z. Tian, C. Ng, and T. Cheng. An $O(n^2)$ algorithm for scheduling equal-length preemptive jobs on a single machine to minimize total tardiness. *J. Sched.*, 9(4), 2006.

[13] M. Xiong, S. Han, and K.-Y. Lam. A deferrable scheduling algorithm for real-time transactions maintaining data freshness. In *RTSS '05*, 2005.