On-line Scheduling of Real-Time Services with Profit and Penalty

Shuo Liu Gang Quan
Electrical and Computer Engineering Department
Florida International University
Miami, FL, 33174
{sliu005, gang.quan}@fiu.edu

Shangping Ren
Computer Science Department
Illinois Institute of Technology
Chicago, IL, 60616
ren@iit.edu

ABSTRACT

In this paper, we study a new family of real-time service oriented scheduling problems. The real time tasks are scheduled non preemptively with the objective of maximizing the total utility. Different from the traditional utility accrual scheduling problem that each task is associated with only a single time utility function (TUF), two different TUFs—a profit TUF and a penalty TUF— are associated with each task, to model the real-time services that not only need to reward the early completions but also need to penalize the abortions or deadline misses. We present two scheduling heuristics to judiciously accept, schedule, and abort real-time services when necessary to maximize the accrued utility. Our extensive experimental results show that our proposed algorithms can significantly outperform the traditional scheduling algorithms such as the Earliest Deadline First (EDF), the traditional utility accrual scheduling algorithms and an earlier scheduling approach based on a similar model.

1. INTRODUCTION

With the proliferation of the Internet has come the opportunity to provide real-time services over the cloud infrastructure [1, 8, 14]. From media on-demand service by Netflix to on-line gaming by Nintendo, from Amazon's e-commerce to Google's free turn-by-turn direction service over the phone, it is fair to say that we are entering a new era of real-time computing. These real-time services are usually built on Internet-based infrastructure, not only because they need to be highly available, but also because they generally rely on large data sets that are most conveniently hosted in large data centers. According to Tim O'Reilly [13], the entire Internet is becoming not only a platform, but also an operating system itself, and "the future belongs to services that respond in real time to information provided either by their users or by nonhuman sensors." [1]

To improve the performance of real-time services, which are mostly soft real-time in nature, one approach is to employ the traditional utility accrual (UA) approach [5, 12]. However, traditional utility accrual (UA) approach [7] uses only one Time Utility Function (TUF) to indicate the task's importance, and most UA scheduling methods (e.g. [10, 11, 15, 16, 17]) imply that the aborted tasks neither increase nor decrease the accrued value or utility of the system.

We believe that, to improve the performance of real-time services over the Internet, it is important to not only measure the profit when completing a task in time, but also account for the penalty when a task is aborted or discarded. In addition, the time at which a real-time service is aborted is also important. First, the more service requests are discarded, or the longer a client waits fruitlessly, the lower the quality of service the client receives. As a result, the service provider has to pay higher cost, either in the forms of monetary compensation or losing future service requests from the unsatisfied client. Second, before a task is aborted or discarded, it needs to consume system sources, including network bandwidth, storage space, and processing power, and thus can directly or indirectly affect the system performance. This is especially true if we assume real-time applications may be dissected and migrated across the entire computing and network infrastructure [4, 9]. Therefore, if a real-time task is deemed to miss its deadline with no positive semantic gain, a better choice should be one that can detect it and discard it as soon as possible.

In this paper, we study the real-time service oriented scheduling problem based on the task model similar to the one proposed by Yu et al. [18]. According to this model, a task is associated with two different TUFs, a profit TUF ($G(t)$) and a penalty TUF ($L(t)$) such that the system takes a profit (determined by its profit TUF) if the task completes by its deadline, and suffers a penalty (determined by its penalty TUF) if the task misses its deadline or is dropped before its deadline. Two non-preemptive scheduling heuristics are presented to optimize the accrual utility when scheduling a set of real-time service requests. The first approach borrows the concept of "opportunity cost" [3] from economics to evaluate the fulfillment of a real-time service request. The second approach employs a more sophisticated but robust method to formulate the potential gain by developing a speculated execution order for the ready tasks. In addition to carefully choosing the ready task to run, our scheduling methods judiciously discard pending requests and abort task executions, and therefore can achieve better performance. Our experimental results show that the proposed algorithms can significantly outperform the traditional scheduling approaches such as the Earliest Deadline First (EDF), the traditional utility accrual scheduling algorithms such as the Generic Utility Scheduling(GUS) [10], the Risk/Reward [6], and a previous scheduling approach based on a similar model, i.e. the Profit Penalty aware scheduling (PP-aware scheduling) [18].

In what follows, section 2 describes the models we used in the paper and presents a motivating example. Section 3 introduces our scheduling approaches in details. Experiment results are discussed

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

in Section 4, and section 5 concludes the paper.

2. PRELIMINARY

The real-time tasks considered in this paper is similar to that proposed by Yu et al. [18]. Specifically, we consider a single sequence of randomly arrived real-time tasks $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, with τ_i defined using the following parameters:

- $[B_i, W_i]$: The best case execution time B_i and the worst case execution time W_i of τ_i;
- D_i: The relative deadline of τ_i;
- $f_i(T)$: The probability density function for the execution time of τ_i;
- $G_i(t)$: The profit TUF, which represents the profit accrued when a task is completed at time t (relative to its arrival time). We assume $G_i(t)$ is a non-increasing unimodal function before its deadline, i.e. $G(t_i) \geq G(t_j)$ if $t_i \leq t_j$, and $G_i(t) = 0$ if $t > D_i$.
- $L_i(t)$: The penalty TUF, which represents the penalty suffered when a task is discarded at time t (relative to its arrival time). We assume $L_i(t)$ is a non-decreasing unimodal function before its deadline, i.e. $L(t_i) \leq L(t_j)$ if $t_i \leq t_j$, and a task is immediately discarded once it missed its deadline.

Note that, even though the deadline of a task can be implicitly defined using appropriate profit and penalty TUFs, we opt to list the deadline explicitly as a parameter for ease of presentation. As shown above, a task is associated with both a profit function and a penalty function with function values varying with time. Therefore, while executing a task has a potential to gain profit, it also has a potential to encounter a penalty at a later time. The system performance is therefore evaluated by its total utility gain after penalty is deducted. Formally, our problem can be formulated as follows.

PROBLEM 1. *Given a task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ as described above, develop on-line, non-preemptive scheduling methods such that the total accrued utility is maximized.*

Problem 1 is NP-hard since a simpler version of this problem, i.e. the total weighted completion time scheduling problem [2], is shown to be NP-hard. To show that the commonly used scheduling policy such as the EDF or the traditional utility accrual approach such as the GUS [10] become ineffective to address this problem, consider the example shown in Figure 1.

Assume that two real-time service requests arrive at the same time ($t = 0$) with their characteristics shown in Figure 1. We assume that the actual processor time of each request is evenly distributed between the interval of its best case and worst case execution time. To make the example more concrete, we assume that the actual processing times for these two requests are 50 and 60, respectively.

When EDF is applied, τ_1 has a higher priority than τ_2 and is executed first. It completes at $t = 50$ with profit of $G_1(50) = 180 - 2 \times 50 = 80$. Then τ_2 starts its execution. At $t = 100$, it misses its deadline and will incur more penalty if its execution continues. Therefore, the execution of τ_2 is aborted at $t = 100$ with penalty of $L_2(100) = 2 \times 100 = 200$. The total utility to process these two requests is therefore $80 - 200 = -120$.

The GUS algorithm chooses the task with the largest potential gain density to execute first. Under our task model, the potential

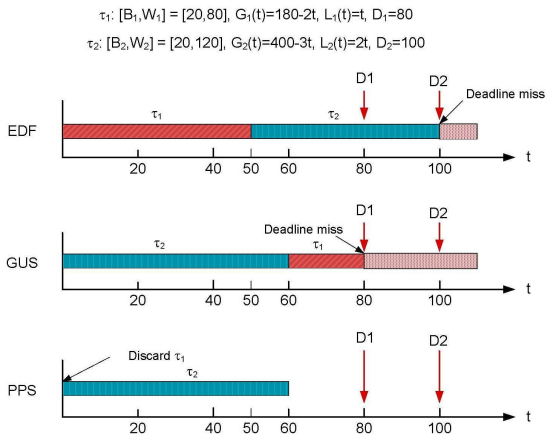


Figure 1: Three different schedules for two real-time tasks τ_1 and τ_2 arriving at the same time $t = 0$.

gain of τ_1 and τ_2, i.e. $\overline{G}(\tau_1)$ and $\overline{G}(\tau_2)$, can be calculated as:

$$\overline{G}(\tau_1) = \int_{20}^{80} (180 - 2t) \times \frac{1}{80 - 20} dt = 80$$

$$\overline{G}(\tau_2) = \int_{20}^{120} (400 - 3t) \times \frac{1}{120 - 20} dt = 190$$

At $t = 0$, we have no knowledge of the actual execution time of τ_1 and τ_2, a reasonable estimate would be the one using their expected values, i.e. 50 and 70, respectively. As a result, τ_2 is chosen to execute first since its potential gain density 190/70 is higher than that of τ_1, i.e. 80/50. It completes at $t = 60$ with profit of $G_2(60) = 400 - 3 \times 60 = 220$. Then τ_1 starts its execution. At $t = 80$, it misses its deadline and is aborted to prevent even higher loss. The total utility to process these two requests is therefore $220 - 80 = 140$.

An astute reader may immediately point out that, after τ_2 completes at $t = 60$, it is less likely that τ_1 can complete by its deadline, given that its best case execution time is 20. Therefore, τ_1 should be immediately aborted at $t = 60$ with a total utility gain of $220 - 60 = 160$. Note that, after τ_2 is selected to execute first, its expected execution time would be 70. Given the expected execution time of τ_1 being 50, it is more likely that τ_1 will miss its deadline. Therefore, a wise scheduling decision would discard it at $t = 0$ with total gain of 220 in this case, as the third schedule shown in Figure 1.

In our example, we can see that the EDF has the worst performance since it makes scheduling decisions solely based on tasks' deadlines. The traditional utility accrual scheduling method takes the individual value function into consideration and therefore can achieve better performance. The problem, however, is that the traditional utility accrual scheduling approaches (such as GUS) fail to take the abortion penalty and the timing for the abortion penalty into consideration. Clearly, how to select the appropriate task to run so as to maximize the profit and how to discard real-time tasks as soon as possible in overloaded situations in order to control the penalty are vital for our research problem.

3. OUR APPROACH

In this section, we present our on-line non-preemptive scheduling solutions to address the problem defined in the previous section. Since the execution of a task may gain positive profit or suffer

penalty and thus degrade the overall computing performance, judicious decisions must be made with regard to executing a task, dropping or aborting a task, and when to drop or abort a task. In what follows, we present two metrics to measure the potential gain when executing a real-time task, and based on which, we develop two scheduling algorithms.

3.1 The opportunity cost based utility metric

Our first utility metric is built upon the concept of *opportunity cost* [3] in economics. In economics, the *opportunity cost* refers to the value associated with the next best available choice that one has to give up after making a choice. When scheduling a set of real-time tasks at $t = T$, let expected gain of running τ_i alone be $\overline{U_i}(T)$ and its opportunity cost be $\overline{OC_i}(T)$. We can define the *system utility* $\widetilde{U}(\tau_i, T)$, i.e. the utility to pick τ_i to run at $t = T$, as

$$\widetilde{U}(\tau_i, T) = \overline{U_i}(T) - \overline{OC_i}(T). \quad (1)$$

The problem becomes how to calculate $\overline{U_i}(T)$ and $\overline{OC_i}(T)$.

Given a task τ_i with its arrival time of r_i, let its predicted starting time be T. Then the expected profit ($\overline{G_i}(T)$) to execute τ_i can be represented as

$$\overline{G_i}(T) = \int_0^{\infty} G_i(t + (T - r_i)) f_i(t | t + T < D) dt \quad (2)$$

$$= \int_{B_i}^{D_i} G_i(t + (T - r_i)) f_i(t) dt \quad (3)$$

Similarly, the expected loss ($\overline{L_i}(T)$) to execute τ_i can be represented as

$$\overline{L_i}(T) = L_i(D) P(t + T > D) \quad (4)$$

$$= L_i(D) \int_{D_i - (T - r_i)}^{W_i} f_i(t) dt. \quad (5)$$

Therefore, the expected utility $\overline{U_i}(T)$ can be represented as

$$\overline{U_i}(T) = \overline{G_i}(T) - \overline{L_i}(T). \quad (6)$$

When $\overline{U_i}(T) > 0$, it means that the probability to obtain positive gain is no smaller than that to incur a loss if we choose to execute τ_i at $t = T$. Since $\overline{G_i}(T)$ is a monotonic decreasing function of T, $\overline{L_i}(T)$ is a monotonic increasing function of T, $\overline{U_i}(T)$ must be a monotonic decreasing function of T. Hence there exists a t_0 such that

$$\overline{U_i}(t_0) = 0. \quad (7)$$

The time $t = t_0$ is called the *critical point*. Apparently, when $t > t_0$, it is more likely that it will incur a loss rather than a gain if we choose to execute τ_i. We can further relax equation (7) by imposing a threshold (δ), i.e.

$$\overline{U_i}(t_0) > \delta. \quad (8)$$

We call δ as the *utility threshold*.

We next introduce how to formulate the opportunity cost when choosing to run task τ_i at $t = T$. The original concept of "opportunity cost" is the value for the next best available choice. It is hard to identify the "next best choice" since the exact reason we need the opportunity cost is to set up the preference order when choosing tasks to run. In our metric, the opportunity cost is calculated as the decay of expected utilities by other tasks. Specifically, let the expected utility of τ_j at $t = T$ be $\overline{U_j}(T)$. Then if we choose τ_i to execute at $t = T$ and after its completion, the expected utility of τ_j is reduced to $\overline{U_j}(T + \overline{C_i})$, where $\overline{C_i}$ is the expected execution time of τ_i. Provided we can remove the task timely when its expected utility is less than zero, we thus define the opportunity cost to run τ_i at $t = T$, i.e. $\overline{OC_i}(T)$, as

$$\overline{OC_i}(T) = \frac{1}{n-1} \sum_{j=1, j \neq i}^{n} \max\left(\overline{U_j}(T) - \overline{U_j}(T + \overline{C_i}), 0\right). \quad (9)$$

With both $\overline{U_i}(T)$ and $\overline{OC_i}(T)$ formulated, we are now ready to introduce our scheduling algorithm. Our scheduling algorithm works at scheduling points that include: the arrival of a new task, the completion of the current task, and the critical point of the current task. The detailed algorithm is described in Algorithm 1.

Algorithm 1 THE SCHEDULING ALGORITHM BASED ON OPPORTUNITY COST

1: **Input:** Let $\{\tau_1, \tau_2, ..., \tau_k\}$ be the accepted tasks in the ready queue, and let $\overline{C_i}$ be the expected execution time of τ_i. Let current time be t and let τ_0 be the task currently being executed. Let the expected utility threshold be δ.
2:
3: **if** A new task, i.e. τ_p arrives **then**
4: Accept τ_p if $\overline{U_p}(\overline{C_0}) > \delta$;
5: Reject τ_p if $\overline{U_p}(\overline{C_0}) \leq \delta$;
6: Remove τ_j in the ready queue if $\overline{U_j}(\overline{C_0}) \leq \delta$;
7: **end if**
8:
9: **if** τ_0 is completed **then**
10: Choose τ_i with the highest system utility, i.e. $\widetilde{U}(\tau_i, t) = \max_k \widetilde{U}(\tau_k, t)$.
11: Remove τ_j in the ready queue if $\overline{U_j}(\overline{C_0}) \leq \delta$;
12: **end if**
13:
14: **if** t = the critical time of τ_0 **then**
15: Abort τ_0 immediately;
16: Choose τ_i with the highest system utility, i.e. $\widetilde{U}(\tau_i, t) = \max_k \widetilde{U}(\tau_k, t)$.
17: Remove τ_j in the ready queue if $\overline{U_j}(\overline{C_0}) \leq \delta$;
18: **end if**

When a new job arrives, its expected utility is calculated based on equation (6). If its expected utility is larger than the pre-set utility threshold, it is accepted, or rejected otherwise. At the same time, the expected utility of the tasks in the ready queue are checked and the task with expected utility less than the threshold is discarded. When the current running task completes, the task in the ready queue with the highest system utility is chosen to be executed. The choice of the new running task will change the expected utility for each task in the ready queue, and the task with expected utility lower than the threshold is discarded. When the time reaches the critical point of the current running task, it implies that it will most likely incur utility less than the utility threshold and is thus worthless of continued execution. In that case, the task is immediately discarded, and a new task will be chosen to be executed. The complexity of Algorithm 1 comes from the calculation of the system utility values for the task set, with the complexity of $O(n^2)$ where n is the number of tasks in the ready queue.

3.2 The speculation based utility metric

From equation (2), (4) and (6), we can clearly see that the potential utility of running a task depends heavily on variable T, i.e. the time when the task can start. If we know the execution order and thus the expected starting time for tasks in the ready queue, we will be able to quantify the expected utility of each task more accu-

rately. In this section, we develop our second utility metric based on a speculated execution order of the tasks in the ready queue.

Algorithm 2 GENERATING THE SPECULATED EXECUTION ORDER AND THE EXPECTED UTILITY FOR TASK IN THE READY QUEUE

1: **Input**:Let $\Gamma = \{\tau_1, \tau_2, ..., \tau_k\}$ be the accepted tasks in the ready queue, and let r_i, $\overline{C_i}$ represent the arrival time and expected execution time of τ_i. Let the current time be t
2: **Output**: The new list $\Gamma' = \{\tau'_1, \tau'_2, ..., \tau'_k\}$ with the speculated execution order and their corresponding expected utility $\widehat{U}(\tau'_j)$ for $\tau'_j, 1 \leq j \leq k$.
3: **if** A task τ_0 is being executed **then**
4: $T = r_0 + \overline{C_0}$;
5: **else**
6: $T = t$;
7: **end if**
8: **while** Γ is not empty **do**
9: **for** Each task C in Γ **do**
10: Calculate $\overline{U_i}(T)$ based on equation (2), (4) and (6);
11: **end for**
12: Select τ_j with the highest $\overline{U_j}(T)$;
13: Add τ_j to the end of Γ';
14: $\widehat{U}(\tau_j) = \overline{U_j}(T)$;
15: $T = T + \overline{C_j}$;
16: Remove τ_j from Γ;
17: **end while**

The general idea to generate the speculated execution order is as follows. We first calculate the expected utility for each task in the ready queue based on the expected finishing time of the current running task. Then the task with the largest one is assumed to be the first task that will be executed after the current task is finished. Based on this assumption, we then calculate the expected utilities for the rest of the tasks in the ready queue and select the next task. This process continues until all tasks in the ready queue are put in order. When completed, we essentially generate a speculated execution order for the tasks in the ready queue and, at the same time, calculate the corresponding expected utility for each task. The detailed algorithm is described in Algorithm 2. The scheduling algorithm based on our speculated utility metric is very similar to Algorithm 1 and is thus omitted. The complexity of the scheduling algorithm mainly comes from Algorithm 2, i.e. $O(n^2)$ with n the number of tasks in the ready queue. In the next section, we investigate and compare the performance of these two algorithms using simulation under a variety of different conditions.

4. EXPERIMENTS

In this section, we use experiments to investigate the performance of our proposed algorithms. The following six representative scheduling approaches were implemented and compared:

- **EDF**: The execution order of the tasks are determined based on the EDF non-preemptive scheduling policy;
- **GUS**: The execution order of the tasks is determined by the potential utility density, or the accrued utility per unit time [10];
- **PP**: This is a previous approach developed based on a metric called *Risk Factor* [18]. It adopts essentially the same system models used in this paper;
- **RR**: The Risk/reward approach described in [6]. This is a utility accrual approach that allows the utility value to be negative;
- **PPOC**: This is the scheduling approach (i.e. Algorithm 1) built upon the utility metric developed based on the opportunity cost;
- **PPS**: This is the scheduling approach built upon the speculated utility based metric as discussed in section 3.2.

4.1 Experiment set up

The test cases in our experiments were randomly generated. Specifically, B, W, and D were randomly generated such that they are uniformly distributed within interval of $[1, 10]$, $[30, 50]$, and $[40, 60]$, respectively. The execution time of a task is assumed to be evenly distributed between interval of [B,W], i.e. $f(t) = \frac{1}{W-B}$. G, L were assumed to be linear functions, i.e. $G(t) = -a_g(t - D)$ in the range of $[0, D]$ and $L(t) = a_l t$. The gradient for $G(t)$ and $L(t)$, i.e. a_g and a_l were randomly picked from the interval of $[4, 10]$ and $[1, 5]$, respectively. Task release times' intervals follow the exponential distribution with $\mu = 5$. The utility threshold δ is set to 0. We conducted four different groups of experiments to study and compare the performance of differen approaches under different conditions. The results are reported as follows.

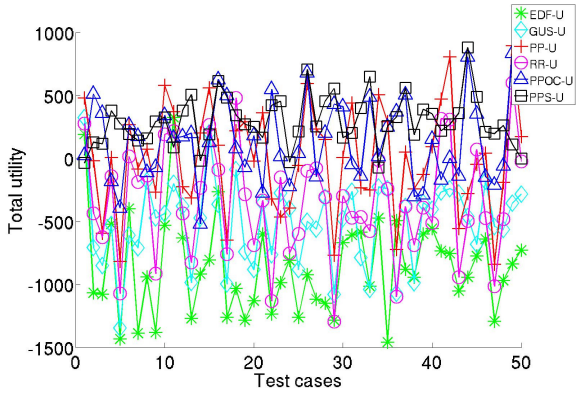
4.2 Experimental results

We first constructed 1000 task sets, each of which consists of ten tasks. The six different scheduling algorithms were applied to these same task sets. The overall utility, the total profit, and the total penalty by each scheduling approach were collected and plotted in Figure 2(a), Figure 2(b), and Figure 2(c), respectively. For ease of presentation, we only show 50 sets of results in the figures.

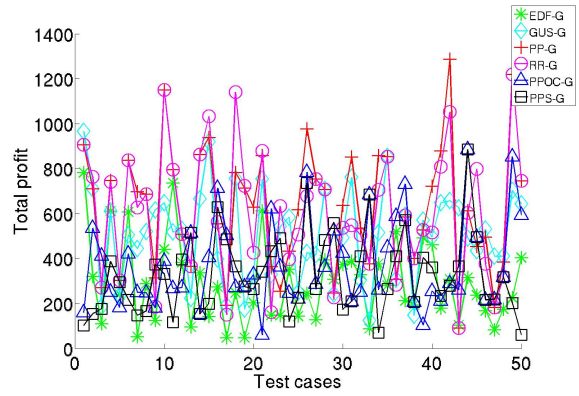
Figure 2(a) clearly shows that both PPOC and PPS can significantly outperform the other scheduling approaches. It is not surprising that, from Figure 2(c), we can see that the penalty-conscious approaches, i.e. PP, PPOC, and PPS, are more effective in controlling the penalty than the other three, i.e. EDF, GUS and RR. PPOC and PPS are particularly effective in penalty control. It is interesting to note from Figure 2(b) and 2(c) that, while the profit by PPOC and PPS are comparable or even inferior to the other approaches, the penalty are dramatically decreased. This is because tasks that would potentially lead to high penalty are declined or discarded at early stages of its execution. As a result, the overall utility are significantly higher than other approaches. As shown in Figure 2(a), with more sophisticated scheduling algorithms to formulate the potential utility more accurately, and thus to make more appropriate decisions in task acceptation, abortion, and discard, PPOC and PPS improve upon PP by more than 120% on average. When comparing PPOC and PPS, we can see from Figure 2(a) that PPS is slightly better than PPOC, but PPS almost dominates PPOC in penalty control.

Next, we study the performance of different scheduling approaches under different workload. In this group of experiments, we varied the number of tasks for each test set from 10 to 50 with an interval of 10, while other parameters remained the same. For each task number, 1000 task sets were generated, and the total utility by each approach were collected and plotted in Figure 3.

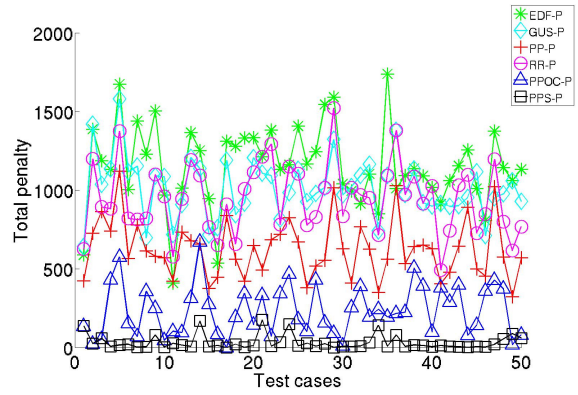
As the number of tasks increases, the ready queue becomes crowded. Figure 3 clearly shows that the performance of EDF, GUS and RR deteriorate quickly due to their poor performance in controlling the penalty. At the same time, we can see clearly from Figure 3 that both PPOC and PPS outperform the other four algorithms. Figure 3



(a) Total utility



(b) Total profit



(c) Total penalty

Figure 2: The comparison of total utility, profit, and penalty by different scheduling approaches.

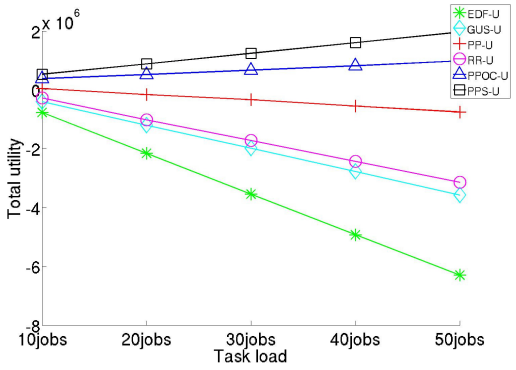


Figure 3: The total utility with different workload.

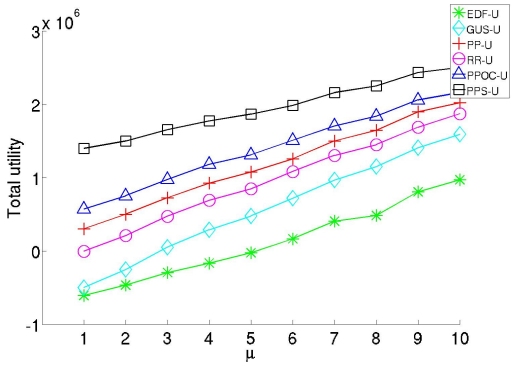


Figure 4: The total utility with different μ.

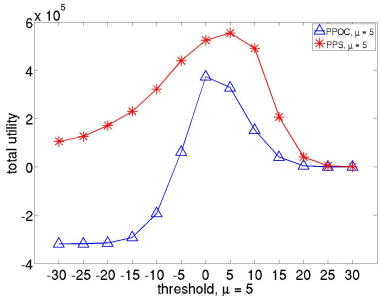
shows the effects on the 1000 task sets' total utility of different values of task load.

We continued to study the scheduling performance under different burstiness conditions. In this experiment, we restored the number of tasks to 10 and varied the task arrival intervals' exponential distribution parameter, μ, from 1 to 10. By changing μ, we essentially changed the intervals between task arrivals. Figure 4 shows the effects on the 1000 task sets' total utility of different values of μ.

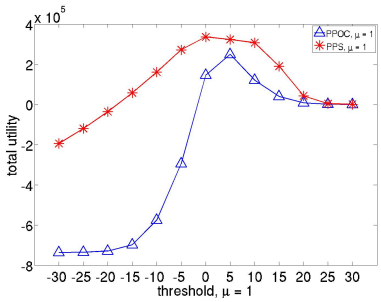
When μ increases from 1 to 10, the number of tasks that comes within the same length of interval decreases and overall workload reduces. From Figure 4, we can see that all the methods have a better performance as μ increases, and PPOC and PPS significantly outperform other approaches.

We further studied the impacts of threshold to the scheduling performance. As indicated in section 3, the threshold plays an important role in task admission, abortion, and execution. The larger the threshold, the smaller the number of tasks can be accepted and executed, and the smaller the penalty the system will suffer. To study this impact, we conducted another set of experiments. We generated test cases as before, but changed the threshold from -30 to 30, with an interval of 5. The total utility were collected and shown in Figure 5. Figure 5 shows the effects on the 1000 task sets' total utility of different values of threshold.

It is interesting to see that the highest utility does not always occur at the point when the threshold equals zero. With the help from the figure, we can tell that the highest utility will seldomly occur at the point with the lowest nor the highest threshold value. The lower the threshold, the more tasks can be accepted to the system and get executed. This helps to improve the value of total profit. However,



(a) Threshold effect with $\mu = 5$



(b) Threshold effect with $\mu = 1$

Figure 5: The total utility varies with the threshold.

having more tasks accepted into ready queue may potentially increase the penalty cost. On the contrary, using a higher threshold helps to control the potential penalty but may limit the total profit that can be obtained. As a result, the total utility is a tradeoff between the two as shown in Figure 5. From Figure 5 we can see the significant impact that the different threshold values may have for the overall performance. In addition, Figure 5 shows that the threshold effect on tasks are different with different tasks' parameters. How to choose an appropriate threshold value for a specific task set to strike the balance between the profit and penalty and hence achieve the optimal accrued utility is an interesting problem and needs further study.

5. CONCLUSIONS

The popularity of the Internet has grown enormously, which has presented a great opportunity for providing real-time services over the Internet. Considering the tremendously large scale of the internet infrastructure, it is necessary that not only the profit but also the cost when executing real-time tasks should be taken into consideration during the resource management process. Our experimental results clearly show that the traditional utility accrued approaches become ineffective in this regard.

In this paper, we present two novel utility accrued scheduling approaches which account for not only the gain by completing a real-time task in time but also the cost when discarding or aborting the task. Our first approach is built upon a metric developed according to the *opportunity cost* concept. The second approach is developed around a speculation-based metric for expected utility. Our scheduling algorithms carefully choose highly profitable tasks to execute, and also aggressively remove tasks that potentially lead to large penalty. Our extensive experimental results clearly show that our proposed algorithms can significantly outperform the traditional EDF approach, the traditional utility accrued approaches, and an earlier heuristic approach based on a similar profit/penalty task model.

6. ACKNOWLEDGEMENT

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, CNS-1018108, CNS-1018731, and CNS-0746643.

7. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. *UC Berkeley*, 2009.

[2] I. D. Baev, W. M. Meleis, and A. E. Eichenberger. Algorithms for total weighted completion time scheduling. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 852–853, 1999.

[3] Z. Bodie, R. Merton, and D. Cleeton. *Financial Economics*. Prentice Hall, New York, 2008.

[4] F. Casati and M. Shan. Definition, execution, analysis and optimization of composite e-service. *IEEE Data Engineering*, 2001.

[5] R. K. Clark. *Scheduling dependent real-time activities*. PhD thesis, Carnegie Mellon University, 1990.

[6] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 160–169, 2004.

[7] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time systems. In *IEEE Real-Time Systems Symposium*, 1985.

[8] E. Knorr and G. Gruman. State of the internet operating system. *http://radar.oreilly.com*, 2010.

[9] H. Kuno. Surveying the e-services technical landscape. In *2nd International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, 2000.

[10] P. Li. *Utility Accrual Real-Time Scheduling: Models and Algorithms*. PhD thesis, Virginia Polytechnic Institute and State University, 2004.

[11] P. Li, H. Wu, B. Ravindran, and E. Jensen. A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints. *Computers, IEEE Transactions on*, 55(4):454–469, April 2006.

[12] C. D. Locke. *Best-effort decision making for real-time scheduling*. PhD thesis, Carnegie Mellon University, 1986.

[13] T. O'Reilly. What cloud computing really means. *O'Reilly Radar, http://www.infoworld.com*, 2010.

[14] A. Weiss. Computing in the clouds. *NetWorker*, 11(4):16–25, 2007.

[15] H. Wu. *Energy-Efficient utility Accrual Real-Time Scheduling*. PhD thesis, Virginia Polytechnic Institute and State University, 2005.

[16] H. Wu, U. Balli, B. Ravindran, and E. Jensen. Utility accrual real-time scheduling under variable cost functions. pages 213–219, Aug. 2005.

[17] H. Wu, B. Ravindran, and E. D. Jensen. Energy-efficient, utility accrual real-time scheduling under the unimodal arbitrary arrival model. In *ACM Design, Automation, and Test in Europe (DATE)*, 2005.

[18] Y. Yu, S. Ren, N. Chen, and X. Wang. Profit and penalty aware (pp-aware) scheduling for tasks with variable task execution time. In *SAC2010 - Track on Real-Time System (RTS'2010)*.