Profit and Penalty Aware Scheduling for Real-Time Online Services

Shuhui Li, Shangping Ren, *Member, IEEE*, Yue Yu, Xing Wang, Li Wang, and Gang Quan, *Senior Member, IEEE*

***Abstract*—As computer and Internet technology continue to advance, real-time online services are emerging. Different from traditional real-time applications for which the scheduling objective is to meet task deadlines, the optimization goal for online service systems is to maximize profit obtained through providing timely services. For this class of applications, there are two distinctive characteristics. First, tasks are associated with a pair of time dependent functions representing accrued profit when completed before their deadlines and accrued penalty otherwise, respectively. Second, the service requests or tasks arrive aperiodically with execution time varying in a wide range. This paper presents a novel scheduling method and related analysis for such applications. Two scheduling algorithms, i.e., the nonpreemptive and preemptive Profit and Penalty aware (PP-aware) scheduling algorithms, are proposed with an objective to maximize system's total accrued profit. Our simulation results clearly demonstrate the advantages of the proposed algorithms, with respect to the system total accrued profit, over other commonly used scheduling algorithms, such as Earliest Deadline First (EDF) and Utility Accrual (UA) algorithms.**

***Index Terms*—Online services, real-time, scheduling.**

I. Introduction

As computers and Internet technology advance, many real-time e-services are emerging [1], [2]. For real-time e-service applications, there is usually a service level agreement (SLA) [3] established between a client and a service provider. Service providers agree to guarantee services with certain levels of quality-of-service, such as timeliness; in return, clients pay for the services completed before their requested deadlines. However, missing a deadline normally does not cause a catastrophic result; instead, penalties are applied if timing requirements are not met [4]–[6]. Furthermore, depending on how fast a request is served, or how soon a client is notified the termination of his/her request due to the inability to meet the required deadlines, the price paid by the client or the penalty suffered by the service provider may be different. To a client, besides the quality of the deliverables, the *timeliness* is a major criterion in judging the quality-of-service provided by the service provider; while to a service provider, the goal is to maximize its long term profits attained under uncertain application workloads and with different classes of SLA constraints [7]–[9].

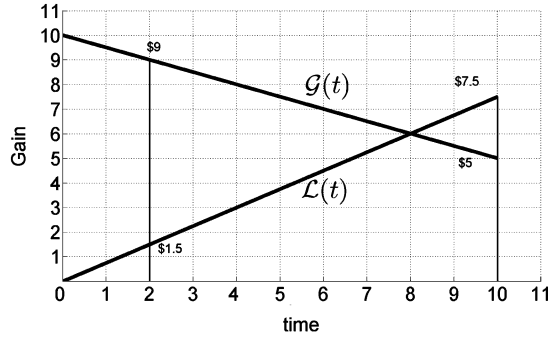

Fig. 1. Time-dependent gain and loss.

For illustrative purpose, consider a real-time e-service request given below.

Example 1: A traveler from city A to city B requests a travel plan from an online travel planning service provider and indicates a deadline (D) he/she is willing to wait. Depending on when the traveler gets the requested travel information from the service provider, the payment is different—the longer it takes, the less he/she pays. In other words, the payment is given only when the request is successfully completed before its deadline and the amount is decided by a nonincreasing time dependent payment function $(\mathcal{G}(t))$. Furthermore, the longer the client waits fruitlessly, the more penalty the system suffers, either by paying more monetary compensation to the client, or losing future service requests from unsatisfied clients. Specifically, the penalty occurs when a request is unfulfilled and it is a nondecreasing time dependent function $(\mathcal{L}(t))$.

To make the example more concrete, we assume that the traveler's deadline is $D = 10$, and the profit and penalty functions are: $\mathcal{G}(t) = -(1/2)t + 10$ and $\mathcal{L}(t) = (3/4)t$, respectively, where $0 < t \leq 10$ for both $\mathcal{G}(t)$ and $\mathcal{L}(t)$. Then, if the request is completed successfully or aborted at time 2, the provider gains \$9 or loses \$1.5, respectively. In contrast, if the request termination happens at the deadline, i.e., time 10, the provider may only gain \$5 if the termination is a successful completion, but lose \$7.5 if the request is aborted at that time. Fig. 1 shows the time-dependent gains and losses. Clearly, if we have sufficient confidence to know that the task would not be completed before its deadline, the earlier we abort the task and notify the client, the less loss the service provider would suffer. □

Manuscript received May 26, 2010; revised December 30, 2010, May 13, 2011, August 17, 2011; accepted September 12, 2011. Date of publication October 18, 2011; date of current version January 20, 2012. This work was supported in part by the National Science Foundation (NSF) under Award CNS-1018731, Award CNS-0746643(CAREER), Award CNS-1035894, Award CNS-0969013(CAREER), Award CNS-0917021, and Award CNS-1018108. Paper no. TII-10-05-0123.

S. Li, S. Ren, Y. Yu, X. Wang, and L. Wang are with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 USA (e-mail: sli38@iit.edu; ren@iit.edu; yyu8@iit.edu; xwang86@iit.edu; lwang64@iit.edu).

G. Quan is with the Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33174 USA (e-mail: gang.quan@fiu.edu).

Digital Object Identifier 10.1109/TII.2011.2172447

At the first glance, it seems that we can directly apply traditional Utility Accrual (UA) approach to solve real-time online service problems stated above. For instance, it would seem that we could use UA-based scheduling algorithms incorporated with the Time Utility Function (TUF) (e.g., [10]–[18]) to schedule the client request for the example above. However, the following critical characteristics inherent in the problem make the traditional scheduling approaches inadequate.

- Different from the traditional UA-based scheduling problems, there are two TUFs that coexist for the same service. Each request is associated with a TUF for profit and a TUF for penalty. Depending on whether the request is successfully completed or aborted, the TUF for profit or the TUF for penalty is applied to compute the system's gain or loss, respectively. Even though the penalty can be defined as negative utility values for tasks aborted or completed after their deadlines, a single variable TUF defined in this way cannot represent both gain and loss caused by completing or aborting a task at different times.
- The processing time for a service may vary widely. For instance, the time to generate a travel plan for driving from city A to B may vary significantly under different road construction scenarios and traffic conditions. Even though we can use the worst-case execution time to schedule each request and ensure the satisfaction of all the accepted requests—as what is usually done in traditional real-time scheduling—such level of conservation is not likely to bring the most profit to service providers.
- Requests arrive aperiodically. For real-time online service applications, tasks arrive aperiodically and system transient overload is a normal rather than an exceptional scenario. Furthermore, often times it is impossible to predict the time interval between two consecutive requests. Traditional UA-aware scheduling algorithms in general assume that tasks are periodic, while scheduling algorithms that do consider aperiodic task arrivals are often not targeted for optimizing utility gains.

In this paper, we introduce a novel scheduling method, i.e., the *Profit and Penalty aware* (PP-aware) scheduling algorithm, that can effectively take these features into account and maximize the system's total accrued profit for real-time online services. Specifically, the main contributions of this paper are listed as follows.

- We introduce a new task model, the *Variable Execution Time with Known Probability*, i.e., the VEP model. This model effectively captures the characteristics of real-time service requests discussed above. At the same time, this model is rather general and can be used to model a large variety of real-time service oriented applications. In this model, task execution time is a random variable ranging from its best-case execution time to its worst-case execution time with a known probability density function. In addition to the common parameters that define a traditional real-time task, i.e., release time and deadline, a task in our model is also associated with two TUFs, i.e., a completion TUF for gains when a task is completed, and an abort TUF for losses when the task is aborted.
- Based on the VEP model, we introduce a novel scheduling policy, i.e., the *Profit and Penalty aware* (PP-aware) scheduling. Two scheduling algorithms, i.e., with and without preemption, are introduced with the objective of maximizing system's total accrued profit.
- We have performed extensive simulations to compare the PP-aware scheduling with other scheduling algorithms in respect of system's total accrued profit under different conditions. Our experiment results show that PP-aware scheduling always outperforms EDF scheduling. While UA-aware scheduling is comparable with the PP-aware scheduling when the system utilization is underloaded, the PP-aware algorithm outperforms the UA-aware algorithm by as much as over 74% in overloaded situations.

The rest of this paper is organized as follows. Section II introduces terms and definitions that are used throughout this paper, and then formally defines the PP-aware scheduling problem. Section III discusses decision making strategies for PP-aware scheduling of a single task. The PP-aware scheduling of multiple tasks without and with preemptions are discussed in Sections IV and V, respectively. Section VI presents our experimental results and discussions. We discuss related work in Section VII and conclude in Section VIII.

II. Preliminaries

In this section, we first introduce several terms used throughout the paper, then formulate a scheduling problem to be addressed.

Definition 1 [Completion Time Utility Function (TUF)]: The *completion TUF* $\mathcal{G}(t)$ for task τ is a *nonincreasing* function of t that denotes the gain accrued when τ is completed at time t.□

Definition 2 (Abort TUF): The *abort TUF* $\mathcal{L}(t)$ for task τ is a *nondecreasing* function of t that denotes the loss accrued when task τ is aborted at time t. □

It is worth pointing out that it takes time to abort a task, especially when the task involves some time-consuming I/O devices. However, although the abort TUF itself does not explicitly define the latency for aborting a service, we can adjust the function value properly to accommodate the additional penalty due to the extra delay.

With both the completion time and abort TUFs, we can define a task's *overall utility function* as follows.

Definition 3 (Overall Utility Function): The overall utility function $\mathcal{U}$ for task τ is defined as

$$\mathcal{U}(t) = \begin{cases} \mathcal{G}(t), & \text{if } \tau \text{ is completed at } t \\ -\mathcal{L}(t), & \text{if } \tau \text{ is aborted at } t \end{cases} \quad (1)$$

where $\mathcal{G}(t)$ and $\mathcal{L}(t)$ are the completion and abort TUF of τ, respectively. □

With functions $\mathcal{G}(t)$ and $\mathcal{L}(t)$, we define VEP task model as follows.

Definition 4 VEP Task Model: A VEP (i.e., Variable Execution Time with Known Probability) task τ is defined by a hex-tuple $(R, [B, W], f(C), \mathcal{G}(t), \mathcal{L}(t), D)$, where R, D, B, W are the task's release-time, deadline, the best-case execution time, and the worst-case execution time, respectively. The $f(C)$ is the task execution time (C) probability density function, $\mathcal{G}(t)$ and $\mathcal{L}(t)$ are task τ's completion and abort TUFs, respectively. □

Note that, the deadline (D) in our case represents the latest time when the service provider needs to abort the task to avoid

large penalty. This deadline can be imposed by the service request itself or based on the expectation of minimal gain set by the service provider. In this paper, we assume $\mathcal{G}(t) > 0$ if $t \leq D$ and the task is automatically aborted at its deadline. Hence, functions $\mathcal{G}(t)$ and $\mathcal{L}(t)$, and thus $\mathcal{U}$, are defined within interval $[0, D]$ only. To simplify the notations and discussions, we introduce an indicator function to identify a domain range.

Definition 5 (Indicator Function): An indicator function $\mathcal{I}_{[a,b]}$ is defined as:

$$\mathcal{I}_{[a,b]}(x) = \begin{cases} 1 & \text{if } x \in [a,b] \\ 0 & \text{if } x \notin [a,b]. \end{cases} \tag{2}$$

□

A VEP task's expected gain at a given time is then defined below.

Definition 6 (Expected Gain): For a VEP task $\tau = (R, [B,W], f(C), \mathcal{G}(t), \mathcal{L}(t), D)$, the expected gain at time t_s is defined as

$$E(\mathcal{G}(T)) = \int_{t_s}^{+\infty} \mathcal{G}(t)\mathcal{I}_{[R,D]}(t)f(t)\mathcal{I}_{[t_s+B,t_s+W]}(t)dt. \tag{3}$$

□

Under the VEP model, a task is associated with both a profit function and a penalty function. Executing a task has the potential to gain profit by completing it in time, it also has a possibility to incur a penalty if it is aborted or the deadline is missed. The overall system performance is therefore evaluated by the total utility obtained when executing all tasks. We formulate the problem to be addressed in this paper as follows.

Problem 1: Given an independent VEP task set $\Gamma = \{\tau_1, \ldots, \tau_i, \cdots, \tau_n\}$, where $\tau_i = (R_i, [B_i, W_i], f_i(C), \mathcal{G}_i(t), \mathcal{L}_i(t), D_i)$, develop online scheduling algorithms that maximize total utility gain, i.e., $\max \sum \mathcal{U}_i(t)$. □

It is not difficult to see that Problem 1 is NP-hard since a simpler version of this problem, i.e., the total weighted completion time scheduling problem is shown to be NP-hard [19]. Therefore, in this paper, we focus on developing an effective heuristic scheduling algorithm to solve this problem.

III. PP-Aware Scheduling of Single VEP Task

Before we introduce the PP-aware scheduling algorithms for multiple tasks in details, we first investigate how a single VEP task is scheduled based on PP-aware scheme. Under the VEP model, task execution time is not known *a priori*. Therefore, before and during the execution of a VEP task, we face choices regarding whether to continue or abort the task. On one hand, the longer we execute the task, the closer we are to the completion point of the task and hence have a high possibility to make profit. On the other hand, due to the nonincreasing property of $\mathcal{G}(t)$ and nondecreasing property of $\mathcal{L}(t)$, the longer the task executes, the less profit we make even if the task finishes before its deadline. We may even have to pay higher loss if it cannot meet its deadline. Hence, in order to maximize the gain for executing a task, a judicious decision as to continue or to abort the task with the consideration of both gain and loss function, i.e., $\mathcal{G}(t)$ and $\mathcal{L}(t)$, is required. This is illustrated in the following example.

Example 2: Consider a VEP task $\tau = (R, [B,W], f(C), \mathcal{G}(t), \mathcal{L}(t), D)$ defined next.

- Release time: $R = 0$.
- Best-case execution time and worst-case execution time interval: $[B,W] = [6, 12]$.
- Task execution time probability density function: $f(t) = (1/6)\mathcal{I}_{[t_0+B,t_0+W]}(t)$, where $\mathcal{I}_{[t_0+B,t_0+W]}$ is a indicator function defined in Section II, Definition 5, and t_0 is the task's starting time ($t_0 = R$ if the system starts executing the task at the time of its release).
- The completion TUF: $\mathcal{G}(t) = (10 - (1/2)t)\mathcal{I}_{[0,10]}(t)$.
- The abort TUF: $\mathcal{L}(t) = (3/4)t\mathcal{I}_{[0,10]}(t)$.
- Deadline: $D = 10$.

We now analyze the system payoffs at different time points. Let T denote the finishing time of τ.

1) At time 0, i.e., when the task is released:
   - if the task is executed, the system's expected gain (defined in Section II Definition 6) is calculated as

$$\begin{aligned} E(\mathcal{G}(T)) &= \int_0^{+\infty} \mathcal{G}(t)f(t)dt \\ &= \int_0^{+\infty} \left(10 - \frac{1}{2}t\right)\mathcal{I}_{[0,10]}(t)\frac{1}{6}\mathcal{I}_{[6,12]}(t)dt \\ &= \int_6^{10} \left(10 - \frac{1}{2}t\right)\frac{1}{6}dt = 4; \end{aligned} \tag{4}$$

   - if the task is aborted at time 0, i.e., not accepted, the penalty function evaluates to $\mathcal{L}(0) = 0$;
   - the probability that the task *cannot* be finished by the deadline is:

$$\begin{aligned} P(T > D) &= 1 - P(T \leq D) \\ &= 1 - \int_0^D f(t)dt = \frac{1}{3} = 33.3\%. \end{aligned} \tag{5}$$

2) At time $9 \in [B, D]$, assume the task is still not finished:
   - if continuing with the task, the conditional expectation of gain given that the task is not finished at 9 is calculated as

$$E(\mathcal{G}(T)|T > 9) = \frac{\int_9^{10}\left(10 - \frac{1}{2}t\right)\frac{1}{6}dt}{P(T > 9)} = 1.75; \tag{6}$$

   - if the task is aborted, the penalty function evaluates to $\mathcal{L}(9) = 6.75$;
   - the conditional probability that the task *cannot* be finished by the deadline is

$$\begin{aligned} P(T > D|T > 9) &= \frac{P(T > D \wedge T > 9)}{P(T > 9)} \\ &= \frac{P(T > 10)}{P(T > 9)} = \frac{2}{3} = 66.7\%. \end{aligned} \tag{7}$$

□

From previous discussions, we can see that at every time instant t, the expected gain and the penalty to abort are different. It is desirable that the decision to abort or continue a task be made judiciously based on these factors. The ratio between a potential loss against an expected gain is hence used as an index to measure the risk of processing a task.

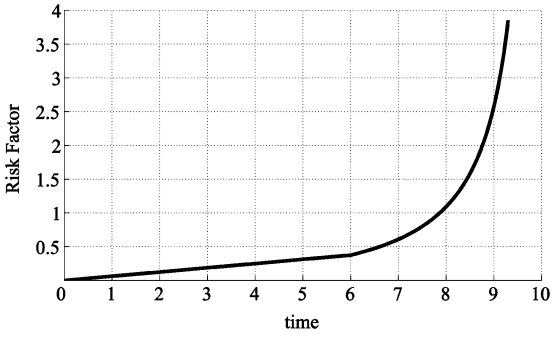

Fig. 2. Risk factor $\rho(t)$.

Definition 7 (Risk Factor): Given a single VEP task, $\tau = (R, [B, W], f(C), \mathcal{G}(t), \mathcal{L}(t), D)$, assume that the task starts at time $t_0 (t_0 \geq R)$ and has not been finished by $t+t_0 (t+t_0 < D)$. Then, the risk factor of task τ at time $t + t_0$ is defined as

$$\rho(t, t_0) = \frac{\mathcal{L}(t + t_0) P(T > D | T > t + t_0)}{E(\mathcal{G}(T) | T > t + t_0)} \tag{8}$$

where T represents the finishing time of τ, and $P(T > D|T > t+t_0)$ and $E(\mathcal{G}(T)|T > t+t_0)$ are calculated by (9) and (10), respectively

$$P(T > D | T > t_0 + t) = \frac{\int f(x)\mathcal{I}_{[t_0+B, t_0+W]}\mathcal{I}_{(D,+\infty)}dx}{\int f(x)\mathcal{I}_{[t_0+B, t_0+W]}\mathcal{I}_{(t_0+t,+\infty)}dx} \tag{9}$$

$$E(\mathcal{G}(T) | T > t + t_0) = \frac{\int \mathcal{G}(x)\mathcal{I}_{[R,D]} f(x)\mathcal{I}_{[t_0+B, t_0+W]}\mathcal{I}_{[t_0+t, D]}dx}{\int f(x)\mathcal{I}_{[t_0+B, t_0+W]}\mathcal{I}_{(t_0+t,+\infty)}dx}. \tag{10}$$

□

From (8)–(10), we have

$$\rho(t, t_0) = \frac{\mathcal{L}(t + t_0)\mathcal{I}_{[R,D]} \int f(x)\mathcal{I}_{[t_0+B, t_0+W]}\mathcal{I}_{(D,+\infty)}dx}{\int \mathcal{G}(x)\mathcal{I}_{[R,D]} f(x)\mathcal{I}_{[t_0+B, t_0+W]}\mathcal{I}_{[t_0+t, D]}dx} \tag{11}$$

As shown in (8), the larger the expected gain, the smaller the *risk factor* is; and the larger the penalty to abort a task, as well as the larger the probability to miss the deadline, the higher the risk factor is to continue the execution.

Fig. 2 graphically illustrates the risk factor function of the task given in the example above with $t_0 = 0$. As can be seen from Fig. 2, when the time approaches the deadline, the risk factor increases dramatically. In fact, when $t \to D$, $E(\mathcal{G}(T)|T > t) \to 0$, and consequently, $\rho(t, t_0) \to +\infty$.

In general, the higher risk a system is willing to take, the higher gain the system may obtain, but at the same time, the loss may also be high as well. Different service providers may be willing to tolerate different risk levels, and only tasks with the risk level lower than the tolerable risk level, i.e., the maximal risk factor $\rho_{\max}$, are accepted and executed.

During the execution of a VEP task (τ) that starts at t_0, if the risk factor at time instant $t_0 + t$, i.e., $\rho(t, t_0)$, exceeds the risk tolerance threshold $\rho_{\max}$, i.e., $\rho(t, t_0) > \rho_{\max}$, and the task is not yet finished, the task is aborted regardless of the amount of time it has been executed. This time instance is called the task's *critical time*.

Definition 8 (Critical Time): Given a maximum tolerable risk factor $\rho_{\max}$, for a task τ that starts at time t_0, its critical time t_{critical} is defined as

$$t_{\text{critical}} = \inf\{t + t_0 : \rho(t, t_0) > \rho_{\max}\}. \tag{12}$$

□

As shown later in the paper, the concept of *critical time* plays an important role in the development of our online scheduling algorithm for Problem 1.

IV. Nonpreemptive PP-Aware Scheduling

Section III reveals several interesting characteristics of PP-aware scheduling of a single VEP task. In this section, we develop our online *nonpreemptive* PP-aware scheduling policy for multiple VEP tasks with the objective of maximizing system accrued utility. The policies consist of an admission test strategy when a new task arrives, and a task selection strategy during the scheduling process.

A. Task Admission Test

Recall that a penalty will incur once a task is accepted but later discarded. The later a task is aborted, the higher the penalty is. Therefore, the purpose of the admission test is to reject a task as early as possible if accepting it is more likely to bring penalty than profit. The question is how to at the *earliest* time *quickly* but not too pessimistically identify tasks that should be rejected. The strategy in our approach is to consider whether at the best favorable scenario, a newly arrived task can contribute profit without adding too much risk. In particular, the best favorable scenario is when all tasks take their best case execution times. Hence, assume a VEP task τ arrives at time t_0. If the potential penalty for each unit of gain when the task takes its best-case execution time B does not exceed the system's tolerable risk level, i.e., if $\rho(B, t_0) \leq \rho_{\max}$, the task is admitted; otherwise it should be rejected. We study the benefit of having the admission test empirically in Section VI-B.

B. Nonpreemptive PP-Aware Scheduling

When VEP tasks are accepted to the pending queue, the problem becomes how to make appropriate scheduling decisions to maximize the system's profit. For nonpreemptive PP-aware scheduling, this scheduling decision is made at the time when:

1) a task successfully completes its execution before its deadline;
2) the current time reaches the critical time (t_{critical}) of the task being executed.

As penalty increases and gain decreases with time, our nonpreemptive PP-aware scheduling method selects a task with the highest expected gain and executes it only to its critical time, aborts and removes a task as soon as it has a higher-than-tolerable risk to cause a loss. Therefore, at each scheduling point t_s, we not only choose the next task to execute, but also further check whether the current selection raises the risks of other tasks beyond the tolerable level, and remove them as soon as possible. Algorithm 1 describes the process of scheduling, where

Algorithm 1: NONPREEMPTIVE PP-AWARE SCHEDULING $(\Gamma, \rho_{max}, t_s)$

```
1  while Γ ≠ ∅ do
2      E_H = 0; τ_H = τ_1; t_action = inf;
3      foreach τ_i ∈ Γ do
4          calculate E(G_i(T)) at scheduling point t_s
           using (3);
5          if E(G_i(T)) > E_H then
6              E_H = E(G_i(T)); τ_H = τ_i;
7          end
8      end
9      calculate τ_H's t_critical using (12);
10     t_action = min{t_critical, D_H};
11     remove task τ_i ∈ Γ that satisfies one of the
       following conditions:
12         • t_s + B_H + B_i > D_i, or
13         • ρ_i(B_i, t_s + B_H) > ρ_max using (8)
14     execute τ_H to min{ τ_H finishing time (t_f),
       t_action};
15     if τ_H does not finish at t_action then
16         abort τ_H;
17         remove τ_H from Γ;
18         t_s = t_action;
19     else
20         t_s = t_f;
21     end
22 end
```

$\mathcal{G}_i(t), \mathcal{L}_i(t), \rho_i(t, t_0)$, and $E(\mathcal{G}_i(T))$ denote task τ_i's gain, loss, risk factor, and expected gain, respectively.

The rationale behind Step 11 in Algorithm 1 is that if we optimistically assume that both τ_H and τ_i take their best-case execution time, task τ_i should be discarded if it cannot meet its deadline. Task τ_i should also be discarded if, upon finishing higher priority task τ_H, the risk level of task τ_i exceeds the system's tolerable risk level.

Time Complexity: The nonpreemptive PP-aware scheduling algorithm is in fact a greedy algorithm. From Algorithm 1, we can see that the complexity of algorithm can be largely formulated as $O(n^2 \times Q)$, where n is the number of tasks to be scheduled, and Q refers to the worst-case computation cost of calculating (3), (8), and (12) for a given task. Note that, as the calculation of formula (3), (8), and (12) does not depend on the number of tasks, i.e., do not depend on n, Q is constant with respect to n. On the other hand, Q depends heavily on the complexity of utility functions $\mathcal{G}(t)$ and $\mathcal{L}(t)$, and task execution time probability density function $f(C)$. For simple linear functions of $\mathcal{G}(t)$ and $\mathcal{L}(t)$ and simple probability density function such as the uniform distribution for $f(C)$, Q can be trivial. However, for a more complicated utility function such as high-order polynomial or nonlinear function, or a more complicated probability density function, Q can be very large.

We next use an example to illustrate Algorithm 1 and also compare it with other traditional scheduling methods such as the EDF and GUS [14].

Example 3: Consider a task set $\Gamma = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_i = (R_i, [B_i, W_i], f_i(C), \mathcal{G}_i(t), \mathcal{L}_i(t), D_i)$:

- $\tau_1 = (0, [4, 40], 1/(40-4), -(t-36), 3t, 36)$,
- $\tau_2 = (6, [3, 36], 1/(36-3), -4(t-42), 3(t-6), 42)$, and
- $\tau_3 = (12, [5, 30], 1/(30-5), -4(t-40), 4(t-12), 40)$

Assume at runtime, the execution time for task τ_1, τ_2, and τ_3 are 20, 12, and 18, respectively. The following cases illustrate the performances of different scheduling algorithms on the task set.

Case 1: EDF: The task execution order is τ_1, τ_3, and τ_2. Completing task τ_1 and gaining $-(20-36) = 16$, and completing task τ_3 and gaining $-4(38-40) = 8$, but missing deadline for task τ_2 and losing $3(42-6) = 108$, therefore, the total accrued utility is -84.

Case 2: GUS: For UA-aware scheduling, we choose the GUS algorithm defined in [14], for which the tasks with largest PUD (potential utility density) [15] is scheduled first. Without loss of generality, we use the average of best-case execution time and worst-case execution time to get the gain per unit of execution for task τ_1, τ_2, and τ_3. They are $36/22$, $4 \times 36/19.5$, and $4 \times 28/17.5$, respectively. Since task τ_1 is released before the other two tasks become available, therefore, the execution order is τ_1, τ_2, and τ_3. Completing task τ_1 and τ_2, but missing deadline for task τ_3, the total accrued utility is -56.

Case 3: Nonpreemptive PP-Aware Scheduling: As the system is overloaded when the last task is released, we set risk level $\rho_{\max} = 1.0$. For task τ_1, we use formula (12) to get the critical time $t_{\text{critical}} = 17$. At time 6 and 12 when task τ_2 and τ_3 are released, we use (11) to calculate the rise factors of executing τ_2 and τ_3, respectively. Both are under the threshold $\rho_{\max}$ which is set to 1, and both are accepted. At time 17, τ_1 is dropped with penalty of 51 and task τ_2 is set to start. Since $\rho_3(5, 17+3) > \rho_{\max}$, τ_3 is discarded with penalty of 20. Task τ_2 is executed to its completion with a gain of 52. The system's total utility gain is thus -19. Fig. 3 depicts the three schedules. □

The above example shows that the PP-aware scheduling algorithm outperforms the traditional approach such as the EDF and GUS. Even though the nonpreemptive PP-aware algorithm (i.e., Algorithm 1) is a greedy approach, it not only takes into account of profit gain $\mathcal{G}(t)$ and loss $\mathcal{L}(t)$, but also other factors such as the risk levels, which helps to improve statistically the performance of the scheduling decisions. In Section VI, we conducted extensive simulation studies to further investigate the performance of the PP-aware nonpreemptive scheduling.

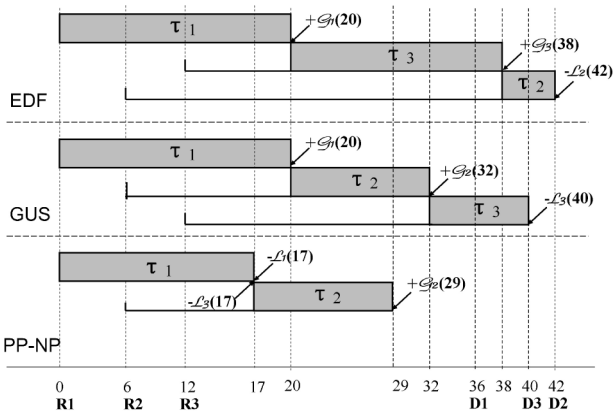

Fig. 3. EDF, GUS, and PP-aware w/o preemption schedule comparison.

V. Preemptive PP-Aware Scheduling

If preemption is allowed and a task with higher potential gain emerges, the current executing task will be preempted by the one with higher gains. As a task's gain and loss functions are time dependent, the expected gains of tasks in the pending queue change as time progresses. Therefore, higher gain tasks may emerge from the pending queue, or arrive from a new client request. This section discusses in detail the preemptive PP-aware scheduling.

A. Preemptive PP-Aware Scheduling

For preemptive PP-aware scheduling, scheduling decisions are made and preemption points are calculated at the time when:

1) a new task arrives;
2) a task successfully completes its execution before its deadline;
3) the current time reaches previously calculated preemption point (Section V-B), or critical time defined in Definition 8, or its deadline, whichever is the earliest.

These time instances are called *scheduling points*. It is worth pointing out that task arrival times are also scheduling points for preemptive PP-aware algorithm which is different from the nonpreemptive PP-aware scheduling algorithm. The reason is that for nonpreemptive PP-aware scheduling algorithm, newly arrived tasks cannot preempt the current task even if they have higher priorities and hence do not call for scheduling decisions; while for preemptive algorithm, current task may be preempted by newly arrivals. Therefore, task arrival times are also scheduling points for the preemptive PP-aware scheduling algorithm.

At each scheduling point t_s, the task with highest expected gain is selected for execution until the next scheduling point. At the same time when the task is selected, its preemption point is also calculated accordingly (see Section V-B). The scheduling algorithm is given in Algorithm 2.

The following example compares the preemptive PP-aware scheduling with EDF and GUS.

Example 4: We use the same example given in Section IV to compare the three preemptive algorithms.

Case 1: EDF: The task execution order is the same as in nonpreemptive EDF. Hence, the total accrued utility is -84.

Case 2: GUS: As task τ_2 has higher PUD than τ_1, task τ_1 is preempted by task τ_2 at time 6. Completing task τ_2, the system obtains 96 gain. As τ_3 has higher PUD than τ_1, τ_3 is executed and the system gains 16. However, as τ_1 misses its deadline, the system loses 108. Hence, the total accrued utility is only 4.

Case 3: PP-Aware: Task τ_1 is executed until task τ_2 is released at time 6. Because τ_2 has higher expected gain than the conditional expected gain of τ_1, τ_1 is preempted. The remaining task of τ_1, i.e., $\tau_1^{\text{remain}} = (0, (0, 40 - 6], 1/34, -(t - 36), 3t, 36)$, is inserted into Γ. When τ_3 arrives, as it has lower expected gain than τ_2, it cannot preempt τ_2. However, at the time when τ_3 arrives, i.e., time 12, as $\rho_1(0, 12) > \rho_{\max}$, task τ_1 is dropped and the system loses 36. τ_2 is executed to its completion with gain of 96. Then τ_3 is selected to run, at time 36, to its completion with gain of 16. The system's total profit is 76. Fig. 4 depicts the three schedules. □

Algorithm 2: Preemptive PP-Aware Scheduling $(\Gamma, \rho_{max}, t_s)$

```
1  while Γ ≠ ∅ do
2      E_H = 0; τ_H = τ_1; t_action = inf;
3      foreach τ_i ∈ Γ do
4          calculate E(G_i(T)) at scheduling point t_s
           using (3);
5          if E(G_i(T)) > E_H then
6              E_H = E(G_i(T)); τ_H = τ_i;
7          end
8      end
9      calculate τ_H's t_critical (12), t_p and τ_p
       (Section V-B);
10     t_action = min{t_critical, t_p, D_H};
11     remove τ_i ∈ Γ that satisfies one of the following
       conditions:
12         • t_s + B_H + B_i > D_i, or
13         • ρ_i(B_i, t_s + B_H) > ρ_max;
14     execute τ_H to min{τ_H finishing time (t_f),
       t_action, a new task τ_h arrival time t_new};
15     if E_τh > E_H then
16         preempt τ_H with τ_h;
17         Γ = Γ ∪ {τ_H^remain};
18         t_s = t_new;
19     end
20     if τ_H is completed at t_f, t_f < t_action then
21         t_s = t_action;
22     else if t_action = t_p then
23         preempt τ_H with τ_p[1];
24         Γ = Γ ∪ {τ_H^remain};
25         t_s = t_action;
26     else
27         abort τ_H;
28         remove τ_H from Γ;
29         t_s = t_action;
30     end
31 end
```

[1]To avoid frequently mutual preempting between two tasks, we set a threshold on the number of preemption times. If the number of preemptions reaches the threshold, preemption is disabled until the current task executed to its critical point, or its completion.

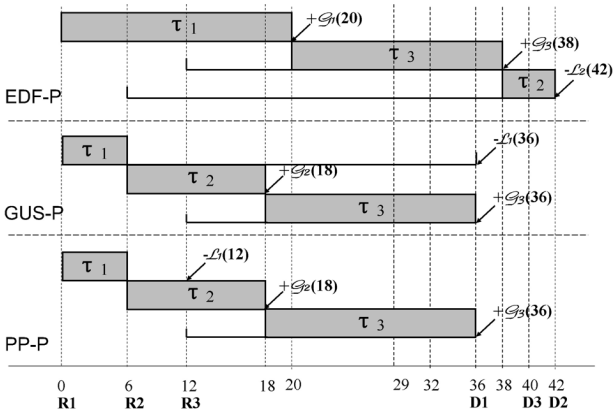

Fig. 4. EDF, GUS, and PP-aware with preemption schedule comparison.

Time Complexity: Similar to Algorithm 1, the preemptive PP-aware scheduling algorithm is also a greedy algorithm and its time complexity can be formulated as $O(n^2 \times (Q + Q_p))$, where n is the number of tasks, Q is the worst case computational cost when calculating (3), (8), and (12) for a given task,

and Q_p is the worst-case computational cost for identifying the potential preemption point for a given task, which is explained further below.

B. Determining the Preemption Point

A key aspect for a preemptive scheduling approach is to determine when preemption should take place.

For a set of VEP tasks $\Gamma = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$, let the current running task be $\tau_c = (R_c, [B_c, W_c], f_c(C), \mathcal{G}_c(t), \mathcal{L}_c(t), D_c)$, $\tau_c \in \Gamma$, which has started at t_{c0}. Assume that at time $t_p (t_p > t_{c0})$, task τ_p's expected gain becomes higher than τ_c. Clearly, time t_p is a legitimated preemption time. The potential preempting task τ_i's expected gain and the current task's conditional expected gain at time t_p are given by (13) and (14), respectively

$$E(\mathcal{G}_i(T)) = \int_{t_p}^{+\infty} \mathcal{G}_i(x)\mathcal{I}_{[R_i,D_i]}(x) f_i(x)\mathcal{I}_{[t_p+B_i,t_p+W_i]}(x)dx \tag{13}$$

$$E(\mathcal{G}_c(T)|T > t_{c0} + (t_p - t_{c0})) = \frac{\int \mathcal{G}_c(x)\mathcal{I}_{[R_c,D_c]}(x) f_c(x)\mathcal{I}_{[t_{c0}+B_c,t_{c0}+W_c]}(x)\mathcal{I}_{[t_p,D_c]}(x)dx}{\int f_c(x)\mathcal{I}_{[t_{c0}+B_c,t_{c0}+W_c]}(x)\mathcal{I}_{(t_p,+\infty)}(x)dx}. \tag{14}$$

Note that since the expected gain of τ_c does not change before the task is executed for at least B_c amount of time, we have $t_p > t_{c0} + B_c$, and therefore,

$$E(\mathcal{G}_i(T)) = E(\mathcal{G}_c(T)|T > t_{c0} + (t_p - t_{c0})) \tag{15}$$

Substitute (13) and (14) into (15), we obtain the constraint (16) for preemption point t_p

$$\int_{t_p}^{+\infty} \mathcal{G}_i(x)\mathcal{I}_{[R_i,D_i]}(x) f_i(x)\mathcal{I}_{[t_p+B_i,t_p+W_i]}(x)dx = \frac{\int \mathcal{G}_c(x)\mathcal{I}_{[R_c,D_c]}(x) f_c(x)\mathcal{I}_{[t_{c0}+B_c,t_{c0}+W_c]}(x)\mathcal{I}_{[t_p,D_c]}(x)dx}{\int f_c(x)\mathcal{I}_{[t_{c0}+B_c,t_{c0}+W_c]}(x)\mathcal{I}_{(t_p,+\infty)}(x)dx}. \tag{16}$$

Based on the definitions of B, W, and D, (16) is rewritten as (17) given below

$$\int_{t_p+B_i}^{\min(t_p+W_i,D_i)} \mathcal{G}_i(x) f_i(x)dx = \frac{\int_{t_p}^{\min(t_{c0}+W_c,D_c)} \mathcal{G}_c(x) f_c(x)dx}{\int_{t_p}^{t_{c0}+W_c} f_c(x)dx}. \tag{17}$$

If (17) has a valid solution t_{p_i} for potential preempting task τ_i, we calculate the task's best-case risk at time t_{p_i}, i.e., $\rho_i(B_i, t_{p_i})$. If the task's best-case risk does not exceed the system's tolerable risk level, i.e., $\rho_i(B_i, t_{p_i}) < \rho_{\max}$, time t_{p_i} is a potential preemption point.

However, as shown by (17), the complexity of computing the potential preemption points depends on the gain function $\mathcal{G}_i$ and task execution time probability distribution function f_i. For complex $\mathcal{G}_i$ and f_i, it may be too difficult to obtain an analytical solution for (17). Under this situation, many iterative approaches, such as the secant method [20], may be used to obtain numerical results.

Fig. 5 shows an iteration process and Table I illustrates the relative errors after each iteration step when the secant method [20] is applied to Example 5. As indicated by Fig. 5 and Table I, the relative error decreases rapidly for each iteration step. For more general scenarios, it has been shown [21], [22] that when the secant method converges to its analytical solution, it converges rather fast, with asymptotic convergence rate of 1.618. That is, at the kth iteration, the error between an approximation value and the analytical solution satisfies (18), where C is a constant. For the given example, the error is only about 1.5% after three iteration steps

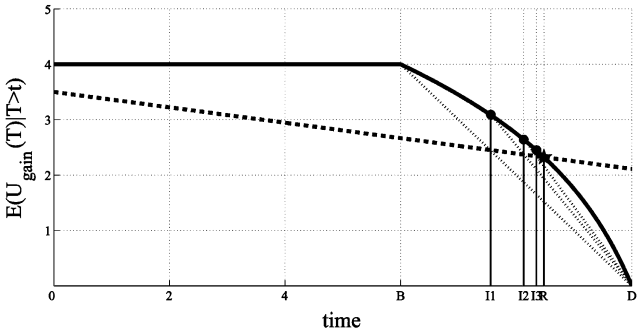

Fig. 5. Iterations used in preemption point computing.

TABLE I
RELATIVE ERRORS VS. ITERATION STEPS

Iteration Steps	1	2	3
Relative error	10.5%	4.1%	1.5%

$$\lim_{k\to\infty} |e_{k+1}| = C|e_k|^{1.618}. \tag{18}$$

After calculating the potential preemption point for each task, the earliest one is set as the valid preemption point and its corresponding task is chosen to preempt the current task at the preemption point. The remaining part of the preempted task, i.e., $(R_c, [\max\{0, B_c - t_p + t_{c0}\}, W_c - t_p + t_{c0}], f_{\text{new}}(T), \mathcal{G}_c(t), \mathcal{L}_c(t), D_c)$, is treated as a new task and inserted into the task queue Γ, where the remaining execution time probability density function $f_{\text{new}}(T)$ is given below

$$f_{\text{new}}(T) = \begin{cases} f_c(t_p + T), & t_p - t_{c0} \le B_c \\ \frac{f_c(t_p+T)}{\int_{t_p}^{t_{c0}+W_c} f_c(x)dx}, & t_p - t_{c0} > B_c \end{cases}. \tag{19}$$

We use an example to illustrate how to calculate preemption point and select the appropriate preempting task.

Example 5: Consider a task set $\Gamma = \{\tau_1, \tau_2\}$, where:

- $\tau_1 = (0, [6, 12], 1/(12-6), -((1/2)t - 10), (3/4)t, 10)$;
- $\tau_2 = (0, [12, 24], 1/(24-12), -((5/36)t - 6), (3/4)t, 35)$.

and we set the system's tolerable risk level $\rho_{\max} = 2$.

Their expected gains at the scheduling point $t_0 = 0$ for task τ_1 and τ_2 are given by $E(\mathcal{G}_1(T))$ and $E(\mathcal{G}_2(T))$, respectively

$$E(\mathcal{G}_1(T)) = \int_0^{+\infty} \left(10 - \frac{1}{2}t\right)\mathcal{I}_{[0,10]}(t)\frac{1}{6}\mathcal{I}_{[6,12]}(t)dt = \int_6^{10} \left(10 - \frac{1}{2}t\right)\frac{1}{6}dt = 4 \tag{20}$$

$$E(\mathcal{G}_2(T)) = \int_0^{+\infty} \left(6 - \frac{5}{36}t\right)\mathcal{I}_{[0,35]}(t)\frac{1}{12}\mathcal{I}_{[12,24]}(t)dt = \int_{12}^{24} \left(6 - \frac{5}{36}t\right)\frac{1}{12}dt = 3.5. \tag{21}$$

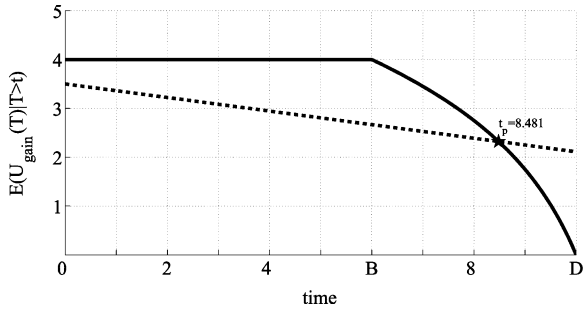

Fig. 6. Example of preemption.

As $E(\mathcal{G}_1(T)) > E(\mathcal{G}_2(T))$, τ_1 is chosen to run at time t_0.

To check if there is any preemption point t_p for τ_2, the expected gain of τ_2 at t_p is given below

$$E(\mathcal{G}_2(T)) = \int_{t_p}^{+\infty} \mathcal{G}_2(t)\mathcal{I}_{[R_2,D_2]}(t)f_2(t)\mathcal{I}_{[t_p+B_2,t_p+W_2]}(t)dt. \quad (22)$$

The expected gain when τ_1 executes to t_p is

$$E(\mathcal{G}_1(T)|T > t_0 + (t_p - t_0)) = \frac{\int \mathcal{G}_1(x)\mathcal{I}_{[R_1,D_1]}(x)f_1(x)\mathcal{I}_{[t_0+B_1,t_0+W_1]}(x)\mathcal{I}_{[t_p,D_1]}(x)dx}{\int f_1(x)\mathcal{I}_{[t_0+B_1,t_0+W_1]}(x)\mathcal{I}_{(t_p,+\infty)}(x)dx}. \quad (23)$$

If there exists a preemption point t_p for task τ_2, (24) shall hold

$$E(\mathcal{G}_2(T)) = E(\mathcal{G}_1(T)|T > t_0 + (t_p - t_0)). \quad (24)$$

As $t_0 + B_1 \le t_p < D_1 \le t_0 + W_1$, i.e., $6 \le t_p < 10 < 12$, (17) has the following two cases.

Case 1: $t_p+B_2 \le D_2 < t_p+W_2$, i.e., $t_p+12 \le 35 < t_p+24$. Hence, $t_p > 35 - 24 = 11$ which is after the deadline of τ_1. Therefore, the t_p is not valid.

Case 2: $t_p + W_2 \le D_2$, i.e., $t_p + 24 \le 35$ In this case, substitute corresponding values into (17), we have

$$\int_{t_p+12}^{t_p+24} \left(6 - \frac{5}{36}t\right)\frac{1}{12}dt = \frac{\int_{t_p}^{10}\left(10 - \frac{1}{2}t\right)\frac{1}{6}dt}{\int_{t_p}^{0+12}\frac{1}{6}dt} \quad (25)$$

i.e.,

$$\frac{7}{2} - \frac{5}{36}t_p = \frac{75 - 10t_p + \frac{1}{4}{t_p}^2}{12 - t_p}. \quad (26)$$

Fig. 6 shows the solution of (26), where the solid line is the conditional expectation of gain of τ_1, i.e., the right side of (26), and the dashed line is the maximum expected gain of τ_2 in terms of time, i.e., the left side of (26). Their intersection point, i.e., $t_p = 8.481$, is the preemption point for τ_2 as it satisfies both $t_p + 24 < 35$ and $\rho_2(12, t_p) < 2$. Therefore, task τ_1 does not finish at time t_p, it will be preempted by τ_2 at that time. □

VI. Experimental Results

In this section, a set of simulations are performed to compare the performance of PP-aware scheduling algorithm with both non-UA-aware (EDF) and UA-aware (GUS) scheduling algorithms in respect to system total accrued gain.

A. Simulation Settings

In our experiments, we assume that both $\mathcal{G}$ and $\mathcal{L}$ are linear functions, and we randomly generate each VEP task $\tau = (R, [B, W], f(C), \mathcal{G}(t), \mathcal{L}(t), D)$ as follows.

- B, W, and D are randomly generated, which are uniformly distributed within [1,10], [30, 50], and [10, 70] respectively, with a constraint $B \le W$.
- The gradient of $\mathcal{G}$ and $\mathcal{L}$, i.e., a_g and a_l, are randomly generated in the range of [4, 10] and [1,5], respectively.
- Task release time are randomly chosen following the Poisson distribution with $\lambda = 5$.
- Functions f, $\mathcal{G}$, $\mathcal{L}$ are defined as follows:
  —-$f(t) = (1)/(W - B)$;
  —-$\mathcal{G}(t) = -a_g(t - D), R \le t \le D$;
  —-$\mathcal{L}(t) = a_l(t - R), R \le t \le D$.

For EDF and GUS scheduling algorithms, if a task is finished at time T before its deadline D, the system gains $\mathcal{G}(T)$ profit from completing the task. On the other hand, if the task misses its deadline, the penalty paid is $\mathcal{L}(D)$.

B. The Benefit of Admission Test

To investigate the benefit of having an admission test, we adopt a concept similar to the "task load" definition given in [23]. In particular, for a given VEP task set, the task load is defined belo.

Definition 9 (VEP Task Load): Given a set of VEP tasks $\Gamma = \{\tau_1, \cdots, \tau_i, \cdots, \tau_n\}$, where $\tau_i = (R_i, [B_i, W_i], f_i(C), \mathcal{G}_i(t), \mathcal{L}_i(t), D_i)$, and they are ordered by their deadlines in the set. At any given time t_0, the system task load is defined below

$$\Gamma_{\text{load}} = \max_{i=1,\ldots,n}\left\{\sum_{j=1}^{i}\frac{\int t \cdot f_i(t)dt}{D_i - t_0}\right\}. \quad (27)$$

□

Under different task loads, we compare system accrued profits when admission test is or is not applied while scheduling tasks based on the nonpreemptive scheduling algorithm (Section IV-B). We randomly generate 100 tasks for each task set with its system load Γ_{load} being 0.2, 0.4, 0.6, 0.8, 1, 1.5, 2, 2.5, 3, respectively, and $\rho_{\max} = 3$. The tests are repeated 100 times and the average over the 100 runs depicted in Fig. 7. As shown in the figure, when the system load is low, the advantage is not significant. This observation confirms with our intuition that under low system load, most of the newly arrived tasks are accepted. However, as the system load increases, the difference becomes more obvious.

C. Performance Comparison Under Different Workload

In our experiments, we randomly generate 100 tasks for each task set with its system load Γ_{load} being 0.2, 0.4, 0.6, 0.8, 1, 1.5, 2, 2.5, 3, respectively. For each system load, 100 task sets are generated and the total expected gains are collected. Fig. 8 shows the average results from the 100 runs for each system load. In this set of experiments, we set $\rho_{\max} = 3$.

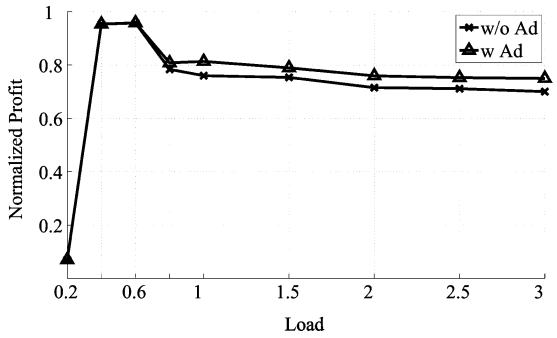

Fig. 7. PP-aware admission test and w/o admission test comparison.

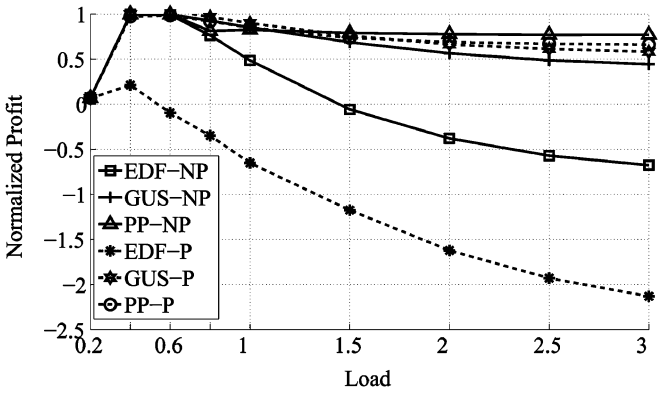

Fig. 8. Performance of different scheduling under different system load.

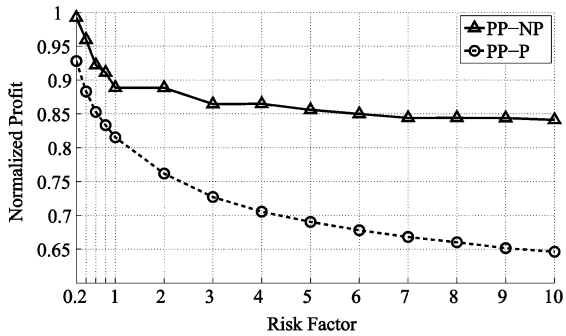

Fig. 9. $\rho_{\max}$ and its impact on overload system performance.

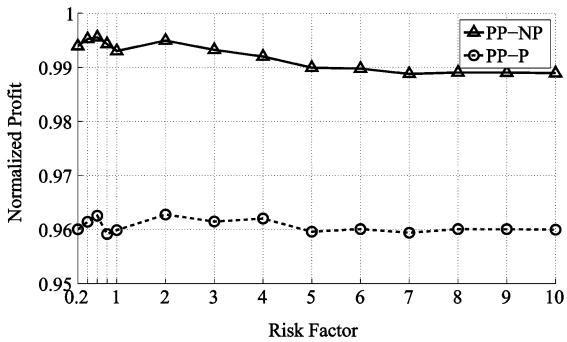

Fig. 10. $\rho_{\max}$ and its impact on underload system performance.

It can be observed from Fig. 8 that when the system load is low (under 40% in the case), all algorithms other than the preemptive EDF scheduling algorithm obtain about the same profit. This is because the preemptive EDF gives the higher priority to tasks with earlier deadlines without considering the system profit. Hence, though under low system load scenario, jobs are able to meet their deadlines, the preemptive EDF may unnecessarily prolong the completion time of tasks with high profit.

Furthermore, for all compared scheduling algorithms, at the beginning, system performances increase as the load increases until they reach their maximals and then start to decrease as the load further increases. The reason behind this observation is that when the system load is small, most of the tasks will meet their deadlines and the system will make more profit if it executes more tasks. However, as the system load increases, some tasks start to miss their deadlines and hence the system starts to encounter penalty. The higher the system load increases, the more tasks may miss their deadline and cause penalty to increase and profit to decrease. In particular, for our test case, when Γ_{load} is around 60%, the system accrued gain under both PP-aware and GUS algorithms reaches its maximum. The profit starts to decrease when the load Γ_{load} increases beyond 60%.

Though the performance of PP-aware and GUS are similar when the system is in underload situation, under high load or overload situation ($\Gamma_{\text{load}} = 3$), nonpreemptive PP-aware scheduling can gain as much as 7541.34, whereas nonpreemptive GUS only gets 4334.06, indicating that PP-aware scheduling algorithm can outperform the GUS scheduling algorithm as much as 74%. The reason is that PP-aware scheduling algorithm takes into account of not only profit gain $\mathcal{G}(t)$ and loss $\mathcal{L}(t)$, but also other factors such as the risk levels, which help to improve statistically the performance of the scheduling decisions.

Another observation is that the performance differences between preemptive and nonpreemptive versions of PP-aware scheduling scheme are small, at most 16%. Similar evidences are observed between preemptive GUS and nonpreemptive GUS as well. The results suggest that if preemption overhead is not negligible, the advantages of preemptive approach for PP-aware scheme become less evident, specially when the complexity of calculating the preemption points is considered.

D. $\rho_{\max}$ and Its Impact

We further study the potential impact that the risk factor, i.e., $\rho_{\max}$, may have toward the performance of the PP-aware scheduling algorithms. For this study, we fix the system load, and only vary the value of the risk factor $\rho_{\max}$. Specifically, we conduct two sets of experiments. In the first set of experiments, we randomly generate 100 tasks for each task set with system load fixed at $\Gamma_{\text{load}} = 3$ (overloaded), and then set $\rho_{\max} = 0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$, respectively. This test is repeated for 100 times and the total gain for each test case is collected. In the second set of experiments, we study the underloaded scenario by setting $\Gamma_{\text{load}} = 0.4$. The average results are shown in Figs. 9 and 10, respectively.

As shown in the Fig. 9, when the system is overloaded, taking lower risk, i.e., $\rho_{\max} = 0.2$, under nonpreemptive PP-aware scheduling, the system gain over 7% more profit than taking higher risk, i.e., $\rho_{\max} = 10$. Similarly, under preemptive PP-aware scheduling, when taking higher risk, the system profit drops 30% comparing with taking lower risk. The results confirm with our intuition that allowing system overload is itself taking a risk when knowing some of accepted jobs may not be finished on time. When preemption is allowed, some jobs which could finish on time may be preempted, and the system have to pay penalties for these jobs. It is obvious that

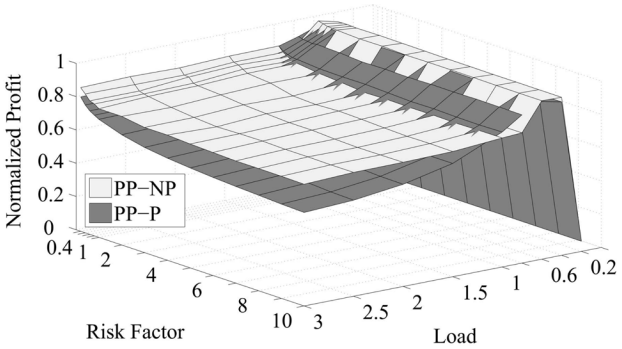

Fig. 11. Comparison of system performance between nonpreemptive PP-aware scheduling and preemptive PP-aware scheduling.

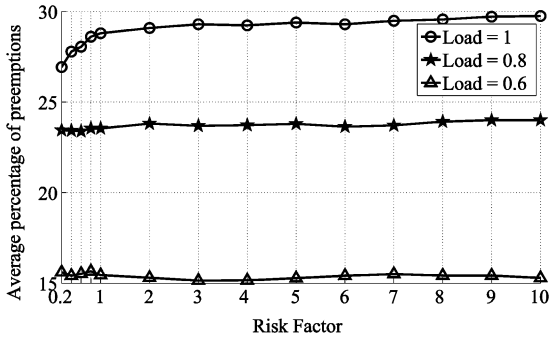

Fig. 12. Preemptions introduced by preemptive PP-aware scheduling as a function of the $\rho_{\max}$.

when the system is overloaded, if the system can not discard nonprofitable jobs on time, it will suffer penalty. The risk factor $\rho_{\max}$ decides how soon the system should react, the smaller value, the more frequent the system has to check if an action should be taken. Hence, under system overload situation, we should use smaller $\rho_{\max}$ for better performance.

On the other hand, when the system is less utilized, the risk factors have less impacts on the system performance, as shown in Fig. 10. The reason is that when the system is underloaded, most tasks will be able to be completed before their deadlines. Although our experimental results show general guidelines on how to choose an appropriate risk factors, how to identify the optimal risk factor for each system is a complex and yet interesting problem and needs further study.

E. Nonpreemptive PP-Aware Scheduling Versus Preemptive PP-Aware Scheduling

Next, we compare the performance of PP-aware nonpreemptive and preemptive scheduling under different workload and risk factors. In this experimental study, we vary both the system load and risk factors. Specifically, we randomly generate 100 tasks for each task set, with system load Γ_{load} set at 0.2, 0.4, 0.6, 0.8, 1, 1.5, 2, 2.5, 3. We also set the risk factors $\rho_{\max}$ to be 0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, respectively. The tests are repeated 100 times and the average over the 100 runs is used in the figures given below.

Fig. 11 compares the system profits between nonpreemptive and preemptive PP-aware scheduling algorithms. Each point on the bright surface is the average system profit using nonpreemptive PP-aware scheduling. The dark surface is the average system profit using preemptive PP-aware scheduling approach. It is worth pointing out that as we use the average system profit instead of actual system profit, it does not reflect individual special situations, such as a performance boost caused by, for instance, a higher profit task comes later than the lower profit tasks.

Fig. 12 shows the average percentage of tasks being preempted introduced by preemptive PP-aware scheduling as a function of $\rho_{\max}$, when system load $\Gamma_{\text{load}} = 0.6, 0.8$ and 1, respectively.

As shown in the Fig. 11, when the system is approaching the threshold of overload, i.e., $\Gamma_{\text{load}} = 1$, preemptive PP-aware scheduling outperforms nonpreemptive PP-aware scheduling. However, as shown in the Fig. 12, on average, about 30% of tasks are preempted. Clearly, such high percentage of task preemption renders the preemption overhead being not negligible. The observation in Section VI-C and the test results in this section indicate that from system profit perspective, nonpreemptive PP-aware scheduling is comparable with preemptive PP-aware scheduling, but it has much less implementation cost and operation overhead. When the system load is in the range of [0.6, 1], which is more common in practice, preemptive PP-aware scheduling outperforms nonpreemptive PP-aware scheduling.

VII. Related Work

For conventional real-time systems, such as process control systems, the major concern is to meet task deadlines. Extensive research has been conducted in designing and analyzing scheduling algorithms for such purposes. The deadline monotonic and rate monotonic scheduling algorithms are among the most widely studied algorithms and form the basis against which other algorithms are compared. However, with these classes of algorithms, the task's *urgency*, rather than its *importance*, is used as job scheduling criteria.

Jensen *et al.* propose to associate each task with a value function to indicate the task's importance [10]. More specifically, a TUF is defined to describe utility accrued by a system at the time when a task is completed. Since then, the topic of UA scheduling, or UA-aware scheduling, has been explored widely in the research community. In particular, in [24], [25], a specific class of TUFs, i.e., step functions, is studied; while [11]–[14] focus on nonstep function of TUFs. Arbitrary shaped TUFs and their corresponding scheduling algorithms are studied by Locke [12] and Li [14].

Wu *et al.* apply UA-aware scheduling on codependent task sets [26], [27] with variable execution times [28]. They introduce the concept of Progressive Utility which generalizes the imprecise computational model and describes the progress of tasks. In this model, utility can be accrued by either finishing current tasks or by improving the progress of related tasks. The computational goal is to maximize the weighted sum of completion time, progressive and joint utilities. Ahmad *et al.* in [29] propose a preemptive utility accrual scheduling (or PUAS) algorithm for adaptive real-time system environment where untoward effects such as deadline misses and overloads are tolerable. The proposed algorithm improves the General Utility Scheduling algorithm by preempting the tasks that GUS abort due to its lower Potential Utility Density (PUD) [15].

In [30], Cortes *et al.* propose a quasi-static scheduling approach, neither purely offline, nor purely online, to find a task execution order that results in maximal utility and at the same time guarantees hard deadlines. With the quasi-static approach, a set of scheduling algorithms are defined at the design time and at runtime, the system can change its scheduling algorithm in order to guarantee hard deadlines and also maximize the accrued utility.

The topic of TUF/UA scheduling is further branched out into other areas, such as in resource sharing [4], [7], [14], [31]–[34], energy consumption [17], [35]–[37], memory cost [38], [39], fault tolerance [40]–[42], overhead [30], and multiprocessor [43], [44] etc.

However, all these existing variations of UA-aware scheduling algorithms share the assumption that utility is accrued only when a task is successfully completed. Although Bartal *et al.* studied the online scheduling problem when penalties have to be paid for rejected jobs [45], [46]. The objective of the proposed algorithm is, nevertheless, to minimize the makespan of the schedule for accepted jobs plus the sum of the penalties of rejected jobs.

In [47], Irwin *et al.* present a heuristics for value-based task scheduling, in which a task's yield decays linearly with its waiting time. However, the penalty only be given if a task is delayed beyond the zero value point.

Liu uses usefulness function in the book [48]. The usefulness of the job becomes zero or negative as soon as the job is tardy. In the latter case, it is *better than late*.

Recently, Niz *et al.* [49] also use the task's criticality (importance) as the scheduling criteria. They consider both scheduling priorities and task criticalities in the Zero-slack scheduling, the task criticalities are similar to the expected gain in our algorithms. It can be assumed that task with higher criticalities will gain higher profit in our system. However, in [49], they assume the task's criticality in the study is static with time and is known before execution. Scheduling criteria based on dynamic criticality, i.e., the task's criticality varies with time and is unknown before execution, is still an open topic.

The PP-aware model and scheduling paradigm are first proposed in [50]. Based on the model, we have extended our work in a number of ways, including using a new metric to quantify the risk when executing a task [51]; developing new scheduling heuristics to maximize the overall profit [52]; and employing PP-aware model to real online service applications [53].

VIII. Summary and Future Work

In this paper, we present: 1) a novel scheduling method and related analysis results for tasks that may not only generate profit when being completed before their deadline, but also incur penalty otherwise and 2) two scheduling algorithms, i.e., the nonpreemptive and preemptive Profit and Penalty aware (PP-aware) scheduling algorithms with an objective to maximize system's total accrued profit. Our simulation results clearly demonstrate the advantages, in respect to system total accrued profit, of the proposed algorithms over other commonly used scheduling algorithms, such as EDF and UA algorithms.

The current work will be extended in a number of ways. First, the PP-aware nonpreemptive and preemptive scheduling algorithms we have proposed in this paper are highly heuristic. Is there an optimal algorithm for the PP-aware scheduling problem we propose in this paper? How close the performance of the proposed nonpreemptive and preemptive scheduling algorithms to the optimal algorithm?

Second, as our experiments have shown, the risk factor $\rho_{\max}$ has an important impact on the performance of the PP-aware scheduling algorithms in system overload conditions. How to identify the optimal $\rho_{\max}$ for different task load is an interesting problem that needs further research.

Third, both the gain and penalty functions in this paper are very simple, i.e., the linear function, we intend to relax this assumption and consider more complicated profit and penalty function. Furthermore, the gain function used in the model represents the "positive gain" the system accrued, it can be extended to represent "negative gain." Similarly, we intend to extend the loss function to represent "negative loss" as well.

Forth, the preemption point calculation is complex and mostly related to the complexity of the profit and penalty functions. If a simple periodic sampling approach is used in replace of calculated preemption point, what is the optimal period and how to adapt when task set changes?

Fifth, our current experiments only compare PP-aware approach with other approaches under the assumption that task execution time is uniformly distributed within a time interval, however, how sensitive the proposed PP-aware scheduling approach is to task execution time distribution model and gain/loss function is yet to be further studied.

Acknowledgment

The authors highly appreciate the constructive comments and suggestions from the reviewers, which contribute significantly to improve the quality of this paper.

References

[1] F. Casati and M. Shan, "Definition, execution, analysis and optimization of composite e-service," *IEEE Data Engineering*, vol. 24, no. 1, pp. 24–39, Mar. 2001.

[2] H. Kuno, "Surveying the e-services technical landscape," in *Proc. 2nd Int. Workshop on Advanced Issues of E-Commerce and Web-Based Inform. Syst.*, 2000, pp. 94–101.

[3] C. S. Yeo and R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing," *Software Practice and Experience*, vol. 36, pp. 1381–1489, 2006.

[4] K. Xiong and H. Perros, "Sla-based resource allocation in cluster computing systems," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2008, pp. 1–12.

[5] L. Zhang and D. Ardagna, "Sla based profit optimization in autonomic computing systems," in *Proc. 2nd Int. Conf. Service Oriented Comput., ICSOC'04*, New York, 2004, pp. 173–182.

[6] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *Proc. 3rd ACM Conf. Electronic Commerce, EC'01*, New York, 2001, pp. 213–223.

[7] D. A. Menasce, V. A. F. Almeida, and M. A. Mendes, "Business-oriented resource management policies for e-commerce server," *Perform. Eval.*, vol. 42, pp. 223–229, 2000.

[8] L. Zhang and D. Ardagan, "Sla based profit optimization in autonomic computing systems," in *Proc. 2nd Int. Conf. Service Oriented Comput.*, 2004, pp. 173–182.

[9] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *Proc. 3rd ACM Conf. Electron. Commerce*, 2001, pp. 213–223.

[10] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time systems," in *Proc. IEEE Real-Time Syst. Symp.*, 1985, pp. 112–122.

[11] K. Chen and P. Muhlethaler, "A scheduling algorithm for tasks described by time value function," *J. Real-Time Syst.*, vol. 10, no. 3, pp. 293–312, May 1996.

[12] C. D. Locke, "Best-effort decision making for real-time scheduling," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1986.

[13] J. Wang and B. Ravindran, "Time-utility function-driven switched Ethernet packet scheduling algorithm, implementation, and feasibility analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 2, pp. 119–133, Feb. 2004.
[14] P. Li, "Utility accrual real-time scheduling: Models and algorithms," Ph.D. dissertation, Virginia Polytechnic Inst. State Univ., Blacksburg, , VA, 2004.
[15] R. K. Clark, "Scheduling dependent real-time activities," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1990.
[16] H. Cho, B. Ravindran, and E. D. Jensen, "Lock-free synchronization for dynamic embedded real-time systems," *ACM Trans. Embedded Comput. Syst.*, vol. 9, no. 3, pp. 23:1–23:28, Feb. 2010.
[17] H. Wu, B. Ravindran, and E. D. Jensen, "Energy-efficient, utility accrual real-time scheduling under the unimodal arbitrary arrival model," in *Proc. ACM Design, Automation, and Test in Europe (DATE)*, 2005, pp. 474–479.
[18] W. T. Strayer, "Function-driven scheduling: A general framework for expressing and analysis of scheduling," Ph.D. dissertation, Univ. Virginia, Charlottesvile, VA, 1992.
[19] I. D. Baev, W. M. Meleis, and A. E. Eichenberger, "Algorithms for total weighted completion time scheduling," in *Proc. 10th Annu. ACM-SIAM Symp. Discrete Algorithms, SODA'99*, 1999, pp. 852–853.
[20] A. Kaw and E. Kalu, "Numerical methods with applications,"1st ed. 2008. [Online]. Available: autarkaw.com
[21] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, "Secant method, false position method, and ridders method," *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, pp. 347–352.
[22] A. Leykin, "Convergence of secant method," 2006. [Online]. Available: http://www.math.uic.edu/leykin/mcs471/NOTES/secant.pdf
[23] G. Buttazzo, *Hard Real-Time Computing Systems*. New York: Springer, 2005.
[24] D. Mosse, M. E. Pollack, and Y. Ronen, "Value-density algorithm to handle transient overloads in scheduling," in *Proc. Euromicro Conf. Real-Time Systems*, 1999, pp. 278–286.
[25] G. Koren and D. Shasha, "D-over: An optimal online scheduling algorithm for overloaded real-time systems," in *IEEE Real-Time Syst. Symp.*, 1992, pp. 290–299.
[26] H. Wu, B. Ravindran, and E. Jensen, "On the joint utility accrual model," in *Proc. 18th Int. Parallel and Distrib. Process. Symp.*, Apr. 2004, p. 124.
[27] H. Wu, B. Ravindran, and E. Jensen, "Utility accrual scheduling under joint utility and resource constraints," in *Proc. 7th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Compu.* , May 2004, p. 307.
[28] H. Wu, U. Balli, B. Ravindran, and E. Jensen, "Utility accrual real-time scheduling under variable cost functions," in *Proc. 11th IEEE Int. Conf. Embedded and Real-Time Comput. Syst. Appl.*, Aug. 2005, pp. 213–219.
[29] I. Ahmad, M. Othman, Z. Zulkarnain, and M. F. Othman, "Improving utility accrual scheduling algorithm for adaptive real-time system," in *Proc. Int. Symp. Inform. Technol., ITSim'08*, Aug. 2008, vol. 1, pp. 1–5.
[30] L. Cortes, P. Eles, and Z. Peng, "Quasi-static scheduling for real-time systems with hard and soft tasks," in *Proc. Design, Autom. Test in Europe Conf. Exhibition*, Feb. 2004, vol. 2, pp. 1176–1181, vol. 2.
[31] A. Popescu, M. Sharaf, and C. Amza, "Sla-aware adaptive on-demand data broadcasting in wireless environments," in *Proc. 10th Int. Conf. Mobile Data Manage.: Syst., Service, Middleware, MDM'09*, May 2009, pp. 142–151.
[32] B. Ravindran, E. D. Jensen, and P. Li, "On recent advances in time/utility function real-time scheduling and resource management," in *Proc. IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput. (ISORC)*, 2005, pp. 55–60.
[33] P. Li, H. Wu, B. Ravindran, and E. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Trans. Comput.*, vol. 55, no. 4, pp. 454–469, Apr. 2006.
[34] H. Wu, B. Ravindran, E. D. Jensen, and U. Balli, "Utility accrual scheduling under arbitrary time/utility functions and multiunit resource constraints," in *Proc. IEEE Real-Time and Embedded Comput. Syste. Appl.*, 2004, pp. 80–98.
[35] H. Wu, "Energy-efficient utility accrual real-time scheduling," Ph.D. dissertation, Virginia Polytechnic Inst. State Univ., Blacksburg, VA, 2005.
[36] H. Wu, B. Ravindran, and E. D. Jensen, "Utility accrual real-time scheduling under the unimodal arbitrary arrival model with energy bounds," *IEEE Trans. Computers*, vol. 56, no. 10, pp. 1358–1371, Oct. 2007.
[37] H. Wu, B. Ravindran, and E. D. Jensen, "Energy-efficient, utility accrual real-time scheduling under the unimodal arbitrary arrival model," in *Proc. Confe. Design, Autom. Test in Europe, DATE'05*, Washington, DC, 2005, pp. 474–479.
[38] S. Feizabadi and G. Back, "Automatic memory management in utility accrual scheduling environments," in *Proc. ISORC*, 2006, pp. 11–19.
[39] S. Feizabadi, B. Ravindran, and E. D. Jensen, "MSA: A memory-aware utility accrual scheduling algorithm," in *Proc. 2005 ACM Symp. Appl. Comput., SAC'05*, New York, 2005, pp. 857–862.
[40] J. Stankovic, "Misconceptions about real-time computing: A serious problem for next-generation systems," *Computer*, vol. 21, no. 10, pp. 10–19, Oct. 1988.
[41] Y. Yu, S. Ren, and O. Frieder, "Prediction of timing constraint violation for real-time embedded systems with known transient hardware fault distribution model," in *Proc. 28th IEEE Real-Time Syst. Symp.*, 2007, pp. 454–466.
[42] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling of fault-tolerant embedded systems with soft and hard timing constraints," in *Proc. Design, Autom. Test in Europ, DATE'08*, Mar. 2008, pp. 915–920.
[43] H. Cho, B. Ravindran, and E. Jensen, "Utility accrual real-time scheduling for multiprocessor embedded systems," *J. Parallel Distrib. Comput.*, pp. 101–110, Feb. 2010.
[44] H. Cho, B. Ravindran, and C. Na, "Garbage collector scheduling in dynamic, multiprocessor real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 6, pp. 845–856, Jun. 2009.
[45] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. S. Gall, and L. Stougie, "Multiprocessor scheduling with rejection," in *Proc. SODA*, 1996, pp. 95–103.
[46] H. Hoogeveen, M. Skutella, and G. Woeginger, "Preemptive scheduling with rejection," *Proc. 8th Annu. Eur. Symp. Algorithms*, pp. 268–277, 2000.
[47] D. Irwin, L. Grit, and J. Chase, "Balancing risk and reward in a market based task service," in *Proc. 13th IEEE Int. Symp. High Perform. Distrib. Comp.*, 2004, pp. 160–169.
[48] J. W. S. Liu, *Real-Time Systems*. New York: Prentice-Hall, 2000.
[49] D. d. Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Proc. 30th IEEE Real-Time Syst. Symp., RTSS'09*, Dec. 2009, pp. 291–300.
[50] Y. Yu, S. Ren, N. Chen, and X. Wang, "Profit and penalty aware (PP-aware) scheduling for tasks with variable task execution time," in *Proc. ACM Symp. Appl. Comput., SAC'10*, New York, 2010, pp. 334–339.
[51] S. Liu, G. Quan, and S. Ren, "On-line scheduling of real-time services for cloud computing," in *Proc. 6th World Congr. Services*, Jul. 2010, pp. 459–464.
[52] S. Liu, G. Quan, and S. Ren, "On-line preemptive scheduling of real-time services with profit and penalty," in *Proc. IEEE Southeastcon'11*, Mar. 2011, pp. 287–292.
[53] S. Liu, G. Quan, and S. Ren, "On-line real-time service allocation and scheduling for distributed data centers," in *Proc. 8th Int. Conf. Services Computing*, 2011, pp. 528–525.

Shuhui Li, photograph and biography not available at the time of publication.

Shangping Ren, photograph and biography not available at the time of publication.

Yue Yu, photograph and biography not available at the time of publication.

Xing Wang, photograph and biography not available at the time of publication.

Li Wang, photograph and biography not available at the time of publication.

Gang Quan, photograph and biography not available at the time of publication.