2011 23rd Euromicro Conference on Real-Time Systems

Scalable Utility Aware Scheduling Heuristics for Real-time Tasks with Stochastic Non-preemptive Execution Intervals

Terry Tidwell[1], Carter Bass[1], Eli Lasker[1], Micah Wylde[2], Christopher D. Gill[1] and William D. Smart[1]
[1]Department of Computer Science and Engineering, Washington University, St. Louis, MO
Email: ttidwell@cse.wustl.edu, {cabass,elasker}@wustl.edu, {cdgill,wds}@cse.wustl.edu
[2]Wesleyan University, Middletown, CT Email: mwylde@wesleyan.edu

***Abstract*—Time utility functions can describe the complex timing constraints of real-time and cyber-physical systems. However, utility aware scheduling policy design is an open research problem. Previously we solved a Markov Decision Process formulation of the scheduling problem to derive value-optimal scheduling policies for systems with periodic real-time task sets and stochastic non-preemptive execution intervals. However, the complexity of computing solutions and their policy storage requirements necessitate the exploration of scalable solutions. In this paper we generalize the Utility Accrual Packet Scheduling Algorithm. We compare several heuristics to Markov Decision Process policy evaluation under soft and hard real-time conditions, different load conditions, and different classes of time utility functions. Based on these evaluations we present guidelines for which heuristics are best suited to particular scheduling criteria.**

I. Introduction

Emerging classes of real-time systems, particularly cyber-physical systems with tightly coupled computational and physical semantics, present special challenges for scheduling policy design. As with traditional real-time systems, satisfying timing constraints is critical to the safe and correct operation of these systems. However, their diverse timing constraints and features such as stochastic non-preemptive execution intervals of tasks fall beyond the capabilities of traditional real-time analysis and design techniques. Time utility functions (TUFs), which map tasks' completion times into measures of the utility of completing them, can be used to represent such diverse timing constraints under a common representation. Utility aware scheduling policies then can be designed with the goal of maximizing system-wide utility accrual. However, the design of such scheduling policies remains an open research topic.

In previous work [1], we proposed a method for off-line design of utility aware scheduling policies for systems with non-preemptable periodic tasks whose job durations are stochastically distributed. By representing these scheduling problems as Markov Decision Processes (MDPs), for each task set we were able to define a measure (called the *value*) of a scheduling policy's expected long term utility accrual with which to choose a policy. However, those techniques are limited by the exponential cost of computing value-optimal scheduling policies.

To address that limitation, in this paper we use a similar method of analysis to evaluate the effectiveness of different on-line scheduling heuristics. The three main contributions of this paper are: (1) *UPA* α and *Pseudo* α, which generalize the Utility Accrual Packet Scheduling Algorithm (UPA) [2] to handle tasks with stochastically distributed non-preemptive execution intervals and arbitrary TUF shapes; (2) an empirical evaluation of relevant on-line heuristics across different classes of TUFs, load demands, and consequences for missing deadlines (i.e., hard versus soft real-time semantics); and (3) guidelines for selecting heuristics under different TUF, load, and deadline criteria, based on that evaluation.

This paper is organized as follows. In Section II we provide a brief survey of related work. Section III describes our system and task model. In Section IV we introduce UPA α and Pseudo α and describe other relevant heuristics that are compared to them in Section V. Section VI offers conclusions including guidelines for using different heuristics, and proposes directions for future work.

II. Related Work

Distributed tracking systems [3] have used utility curves to describe the value of associating raw radar data with active tracks as a function of time. Time utility functions are also well suited for use in control systems. Control loops may become unstable due to inter-job jitter [4]. Time utility functions can describe sensitivity to this jitter to better schedule jobs such as actuation and sensing [5]. Utility also has been proposed as a way to schedule communication traffic in control area networks in order to guarantee cyber-physical properties such as cruising speeds in automobiles [6].

The concern addressed in this paper - scheduling tasks with stochastic non-preemptive execution intervals - is especially relevant in distributed control networks, where it is undesirable to preempt messages already on a CAN and where network delays may be unpredictable. Similar scheduling problems also may occur in real-time systems built from COTS peripherals, where access to the I/O bus may need to be scheduled in order to guarantee real-time performance [7].

There are several existing techniques for addressing stochastic system behavior in real-time systems. Statistical Rate Monotonic Scheduling [8] deals with periodic tasks with

1068-3070/11 $26.00 © 2011 IEEE
DOI 10.1109/ECRTS.2011.30

stochastic durations. Constant Bandwidth Servers (CBS) [9] guarantee resource availability in the face of sporadic tasks or tasks with stochastic durations.

Stochastic analysis of Constant Bandwidth Servers and other schedulers has used techniques similar to those presented in this paper. Examples include calculating the probabilistic response times for interrupt scheduling using Constant Bandwidth Servers [10] and analysis of different Constant Bandwidth Server parameters in mixed hard/soft real-time settings in order to perform distributed scheduling [11]. Analysis of scheduling policies for non-preemptive tasks with stochastic duration [12] has focused on calculating the expectation of a different scheduling metric (deadline miss rates) as opposed to utility accrual. Stochastic analysis also has been applied to global multiprocessor scheduling to calculate expected tardiness for soft real-time systems [13].

Stochastic analysis techniques such as Markov Decision Processes (MDPs) can be used not only to perform analysis, but for design. MDPs are used to model and solve sequential decision problems in cyber-physical domains such as helicopter control [14] and mobile robotics [15], [16]. In previous work we applied these techniques to generating share [17]–[19] and utility [1] aware scheduling policies for scheduling tasks with stochastic non-preemptive execution intervals.

Several specialized utility aware scheduling techniques have been proposed, including the Dependent Activity Scheduling Algorithm [20], the Utility Accrual Packet Scheduling Algorithm (UPA) [2] which extends previous work by Chen and Muhlethaler [21], and the Gravitational Task Model [5]. Each makes assumptions about the shapes of the time utility functions that do not hold in general. However, as we show in Section IV, the UPA algorithm can be extended in order to work in the scheduling domains explored by this paper. Other methods for general utility aware scheduling [1], [22]–[24] are the basis for the comparisons presented in this paper. They are also discussed in detail in Section IV.

III. System Model

As in our previous work [1], the system model is assumed to have n non-preemptable periodic tasks, denoted $(T_i)_{i=1}^{n}$, which are in contention for a shared resource. Each task is composed of a series of jobs, where $J_{i,j}$ refers to the jth job of task T_i. The period of task T_i, denoted p_i, is the inter-arrival time between the release time $r_{i,j}$ of job $J_{i,j}$ and the release time $r_{i,j+1}$ of job $J_{i,j+1}$. Released jobs stay in a ready queue until they are scheduled to use the resource, or they expire due to no longer being able to earn utility.

A scheduler for the shared resource repeatedly chooses either to run a job from the ready queue or to idle the resource for a quantum. As we discuss further in [1], the ability to idle the resource, (i.e. to be non-work-conserving) may be crucial for a scheduling policy to prevent *target sensitive* tasks (whose utility is highest at a target time) from completing too early. When chosen to run, a job is assumed to hold the resource for a stochastic duration, the distribution of which is captured by the probability mass function D_i. This function is assumed to have support on the range $[1, w_i]$, where w_i is the maximal (referred to as the *worst case*) execution time for any job of T_i and $D_i(t)$ is the probability that a job of T_i runs for exactly t quanta. If job $J_{i,j}$ is completed at time t, utility is earned as denoted by the function $U_i(t)$. This function is assumed to have support on the range $[1, \tau_i]$, where $r_{i,j} + \tau_i$ is the time at which the job expires and is removed from the ready queue. $U_i(t)$ defines how much utility is gained by completing $J_{i,j}$ at time $r_{i,j} + t$.

If the job expires before it completes, the system instead is assessed a penalty e_i. The goal is to maximize the difference between the total system wide utility accrual and the sum of the penalties accrued by expiring jobs. This system model is widely applicable and can model hard real-time deadlines [1] by setting τ_i to the deadline and e_i to ∞. Thus, both soft real-time and hard real-time tasks can be mixed within a single task set under our approach. Although not required by the system model we assume that τ_i is in the range $(w_i, p_i]$ for the analysis and experiments presented in this paper. The constraint $\tau_i \leq p_i$ bounds the size of the run queue to n jobs, with only one job per task. The constraint $w_i < \tau_i$ ensures that a job can complete without penalty if given the resource immediately upon release.

IV. Solution Approach

In previous work [1] we showed how utility accrual scheduling problems for non-preemptive tasks with stochastic execution times could be represented under our system model as Markov Decision Processes (MDPs). This allows us to do two things: (1) given a scheduling policy it allows us to calculate the *value* gained by running that policy, which is a measure of a policy's quality; and (2) given a task set within the system model described in Section III it allows us to calculate a value-optimal policy, which is a scheduling policy that maximizes the expected value measurement over long term execution.

In Section IV-A we first provide a brief overview of how a scheduling MDP is constructed and how the corresponding policy value function is calculated. Sections IV-B through IV-D describe relevant scheduling heuristics involving permuting the ready queue (sequencing heuristic), maximizing immediate reward (greedy heuristic), or using deadlines to maximize utility (deadline heuristic). Section IV-E describes the Utility Accrual Packet Scheduling Algorithm (UPA) [2] and presents new heuristics UPA α and Pseudo α, which extend UPA to deal with: (1) stochastic execution intervals, (2) arbitrary time utility function (TUF) shapes, and (3) potential costs and benefits of permuting the ready queue. In Section V we present an empirical evaluation of these five heuristics across different classes of TUFs, load demands, and penalties for missing deadlines.

A. Scheduling Markov Decision Process

An MDP is a five-tuple $(\mathcal{X}, \mathcal{A}, P, R, \gamma)$. Our scheduling MDP is defined as a transition system over a set of scheduler states, $\mathcal{X}$. Each state has two components: a variable t_{system} which tracks the time passed since the beginning of the task

hyperperiod (the least common multiple of the task periods), and an indicator variable for each task $q_{i,j}$, which tracks whether task T_i has a job in the ready queue that needs to be scheduled. Because in general tasks may have multiple jobs in the ready queue, jobs are indexed so that the most recently released job of T_i is tracked by the variable $q_{i,1}$. Because jobs expire, only $\lceil \tau_i/p_i \rceil$ variables are needed to track all the jobs of T_i that can be in the ready queue at one time.

A set of scheduling actions, $\mathcal{A}$, defines the different decisions the scheduler can make. Action $a_{i,j}$ is the scheduling decision that runs the job mapped to $q_{i,j}$, and action a_{idle} is the scheduling decision that simply idles the resource for one quantum. The transition function $P(y|x,a)$ defines the probability of moving from state x to state y given that action a was taken. This function is dependent on the duration distribution for the task scheduled. A policy π maps each state to an action that the scheduler will take in that state. These policies may be precomputed, or the state may be given as input to a scheduling decision function at runtime.

The final two components of the scheduling MDP are the reward function R and the discount factor γ. Together, these two components define the value function, which is the quality metric used to compare scheduling policies. The state of a system evolves according to its scheduling policy over a series of decision epochs (intervals between policy decisions), such that immediate reward may be gained after each decision. The immediate reward r_k gained from moving from state x to state y after decision epoch k when action a is taken, is given by the reward function $R(x,a,y)$. This is defined to be the utility gained (if any) by the scheduled task, while c_k is defined to be the sum of any penalties incurred by expiring tasks over that decision epoch. Thus the value of a policy π in a deterministic setting, i.e., $\forall x,y,a \ P(y|x,a) \in \{0,1\}$, is the sum of the infinite series:

$$V^{\pi} = \sum_{k=0}^{\infty} r_k - c_k \tag{1}$$

However, this value does not take into account the length of each action. To account for the fact that actions may have different durations, we define t_k to be the time that the scheduled job took to execute. The value of a policy is then:

$$V^{\pi} = \sum_{k=0}^{\infty} r_k/t_k - c_k \tag{2}$$

This allows us to differentiate between actions of different lengths. The value r_k/t_k is called the utility density of the scheduling decision. Because no restriction is put on the shape of the TUF, $r_k/t_k - c_k$ can take on arbitrary value and this sum can diverge to infinity. In order to prevent this divergence the discount factor, γ is used:

$$V^{\pi} = r_0/t_0 - c_0 \sum_{k=1}^{\infty} \gamma^k r_k/t_k - c_k \tag{3}$$

This discount factor is set in the range $[0,1]$ and is typically set close to 1. The policy values presented in this paper use $\gamma = 0.99$. A discussion of this discount factor can be found in [25], but the intuition for this choice is that it offers a compromise between being close enough to 1 to capture long term policy effects, while still allowing MDP solving algorithms to converge to a solution reasonably quickly.

For deterministic systems and a fixed policy, only one state evolution is possible, with only one sequence of rewards. However, in our domain actions have stochastic duration. To account for this the value function becomes the expectation of the set of infinite sequences possible under the scheduling policy, with each sequence weighted by its likelihood and the value for the policy being the weighted average sum.

This policy evaluation method has two main advantages. First, it provides a deterministic measure of quality in the face of stochastic behavior. The value calculated by this methodology is the same value that an actual run of the system will have in the limit, over the long term. Second, it accounts for low probability high impact events in calculating the policy value. Unlike quality measures derived from Monte Carlo simulation which may miss a rare event, by taking the expectation from all possible state evolutions the value of a policy quantifies effects of rare high impact events. This is especially relevant when the penalty associated with a job expiring, e_i, is large compared to the possible utility values.

This formulation as an MDP allows us to derive a *value-optimal* scheduling policy [1] that maximizes the value function for a given discount factor. However, this schedule must be pre-computed, and the resulting computation and storage costs can make doing so intractable. To calculate a policy the full state space of the system must be enumerated, and the optimal value function calculated using modified policy iteration [25]. The cost of doing so is polynomial in the size of the state space, which is in turn exponential in the number of tasks. For use at run-time, the value-optimal policy must then be stored in a look up table, the size of which could be as large as the size of the state space. Because of this, we are interested in how well heuristics with lower computation and storage costs can approximate the value-optimal policy under different system scenarios. In the rest of this section, we describe several relevant heuristics, whose performances are compared to that of the value-optimal policy in the experimental evaluation presented in Section V.

B. Sequencing Heuristic

A straightforward (if relatively expensive) approach to producing utility-aware schedules is to calculate exhaustively the expected utility gained by scheduling every permutation of the jobs in the ready queue. If we assume $\tau_i < p_i$ for all tasks, there are $n!$ permutations of the ready queue, and the calculation of expected utility requires convolving the duration distributions of each job in the sequence, an operation that takes time proportional to the maximum task execution time. While impractical for deployment as a scheduler, this heuristic is optimal in the sense that given no future job arrivals, no

work conserving schedule can gain more expected utility. Thus this policy is a good benchmark for measuring what is gained by considering future job arrivals and being non-work conserving (like the value-optimal schedule). We refer to this policy as the *sequencing heuristic*.

C. Greedy Heuristic

The *greedy heuristic* is equivalent to the value-optimal schedule if the discount factor γ is set to zero. When this happens the scheduler only considers the effect of the expected immediate reward, and not the long term impact of the scheduling decision. Calculating this policy in the soft real-time case, where all penalties for expiring jobs e_i are set to zero, is $O(n)$ because the expected immediate reward for each scheduling action can be precomputed and is independent of what other jobs are in the ready queue. In the hard real-time case the immediate reward is not independent of the other jobs, because these other jobs may have a non-zero probability of expiring depending on the stochastic duration of the job under consideration. Calculating this conditional expectation of the expiration cost requires time $O(w)$ where w is the worst case execution among all tasks in the system. Since this must be done for every task, the total complexity is $O(nw)$, which may be unacceptably high. Using potential utility density as a heuristic for utility-aware scheduling was proposed in [22], [23]. The Generic Benefit Scheduler [24] uses this heuristic to schedule chains of dependent jobs according to which chain has the highest potential utility density.

D. Deadline Heuristic

The Best Effort Scheduling Algorithm [23] uses Earliest Deadline First scheduling [26] as a basis for utility-aware scheduling. Deadlines are assigned to tasks based on the task's time utility function. Although optimal deadline placement is an open problem, deadline assignment is typically done at critical points in the task's time utility function, where there is a discontinuity in the function or in its first derivative. We refer to this scheduling algorithm as the *deadline heuristic*.

E. UPA α and Pseudo α

The Utility Accrual Packet Scheduling Algorithm (UPA) [2] uses a *pseudoslope* heuristic to order jobs based on the slope of a strictly linearly decreasing approximation of the task's time utility function. This algorithm was developed for use in systems with non-increasing utility functions and deterministic execution times. UPA first selects the set of jobs that will finish before their expiration times, then sorts the rest of the jobs by their pseudoslope (given by $-U_i(0)/\tau_i$). The slope closest to negative infinity is placed first in the calculated schedule. Finally UPA does a bubble sort of the sorted jobs in order to find a locally optimal ordering, in a way similar to the sequencing heuristic discussed in Section IV-B.

To account for time utility functions with arbitrary shapes (as opposed to strictly decreasing time utility functions) our first extension to UPA is to calculate the pseudoslope value using the current value of the utility function at the time when the scheduling decision function is invoked. At time $r_{i,j}+t$, for instance, the pseudoslope value for job $j_{i,j}$ is $-U_i(t)/(\tau_i - t)$.

To make UPA applicable to tasks with stochastic durations, we introduce a parameter α in $[0, 1]$ that imposes a minimum threshold on the probability that the job will finish before its termination time. In general *UPA* α considers only jobs whose probability of timely completion is greater than or equal to α, and in particular UPA 0 considers any job while UPA 1 considers only those jobs that are guaranteed to finish before their deadlines.

Finally, with deterministic durations the local search for a better scheduling order is $O(n^2)$, but with stochastic durations convolutions of the duration distributions also need to be calculated to find the expected utility of each sequence of jobs. While calculating the expected value of a sequence after an inversion of two jobs in a sequence is $O(1)$ in the deterministic case, in the stochastic case it is $O(w)$. This makes this part of the UPA algorithm potentially expensive for online scheduling use. To evaluate the cost and benefit of this sequencing step we define two distinct heuristics: Pseudo α is the scheduling algorithm that simply uses the pseudoslope ordering to schedule jobs, while UPA α performs the additional sequencing step to find a locally optimal schedule.

V. Experimental Evaluation

We evaluated the heuristics described in Section IV using three different classes of time utility functions. A *downward step* utility curve is parameterized by the task's termination time τ_i and utility upper bound u_i and is defined as:

$$U_i(t) = \begin{cases} u_i & : \quad t < \tau_i \\ 0 & : \quad t \geq \tau_i \end{cases} \tag{4}$$

This family of functions is representative of jobs with firm deadlines, like those considered in traditional real-time systems. For the purpose of the deadline heuristic, job deadlines are assigned at the point τ_i. Unlike approaches that consider only deadlines, the relative utility upper bounds are equally important in determining good utility aware schedules.

A *linear drop* utility curve is parameterized in the same manner as a downward step utility curve, but with an additional parameter describing the function's critical point c_i:

$$U_i(t) = \begin{cases} u_i & : \quad t < c_i \\ u_i - (t - c_i)\frac{u_i}{\tau_i - c_i} & : \quad c_i \leq t < \tau_i \\ 0 & : \quad t \geq \tau_i \end{cases} \tag{5}$$

Such a utility function is flat until the critical point, after which it drops linearly to reach zero at the termination time. This family of curves is representative of tasks with soft real-time constraints, where quality is inversely related to tardiness. For that reason the deadline is set to the point c_i.

A *target sensitive* utility curve is parameterized exactly like a linear drop utility curve, and is defined as:

$$U_i(t) = \begin{cases} t\frac{u_i}{c_i} & : \quad t < c_i \\ u_i - (t - c_i)\frac{u_i}{\tau_i - c_i} & : \quad c_i \leq t < \tau_i \\ 0 & : \quad t \geq \tau_i \end{cases} \tag{6}$$

The utility is maximized at the critical point, and is representative of tasks whose execution is sensitive to inter-task jitter. In control systems, whose sensing and actuation tasks are designed to run at particular frequencies, quality is inversely related to the distance from the target point.

For the experiments presented in this paper, τ_i is chosen uniformly at random from the range $(w_i, p_i]$. Because the termination time precedes the next job's inter-arrival only one job of a task is available to run at any given time. Because the termination time is greater than the worst case execution time, the task is guaranteed to complete if granted the resource at the instant of release. The upper bound on the utility curve u_i is chosen uniformly at random from the range $[2, 32]$, and the critical point c_i for the target sensitive and linear drop utility curves is chosen uniformly at random from the range $[0, \tau_i]$.

Task periods were randomly generated to be divisors of 2400 in the range $[100, 2400]$, ensuring the hyperperiod of the task set was constrained to be no more than 2400.

The duration distributions for each task were parameterized with three variables (l_i, b_i, w_i) such that $l_i \leq b_i \leq w_i$ where l_i and w_i are the best case and worst case execution times respectively, and that 80% of the probablity mass is in the range $[l_i, b_i]$. The duration distribution is defined as:

$$D_i(t) = \begin{cases} 0 & : \quad t < l_i \\ \frac{0.8}{l_i - b_i} & : \quad l_i \leq t \leq b_i \\ \frac{0.2}{b_i - w_i} & : \quad b_i < t \leq w_i \\ 0 & : \quad w_i < t \end{cases} \tag{7}$$

This means that the demand of the task is normally between l_i/p_i and b_i/p_i but occasionally may be as high as w_i/p_i. The parameters are further constrained such that $l_i/p_i \geq 0.05$ and $b_i/p_i \geq 0.10$.

In addition to the constraints on the individual tasks, constraints are set on the task set as a whole, such that:

$$\sum_{i=1}^{n} l_i/p_i = L_i$$

$$\sum_{i=1}^{n} b_i/p_i = B_i$$

$$\sum_{i=1}^{n} w_i/p_i = W_i$$

where the 3-tuple (L_i, B_i, W_i) defines the overall system load. The nominal case assumes the values $(0.70, 0.90, 1.20)$, which we call the high load scenario, where the resource is working near capacity with transient overloads. A more conservative case, the medium load scenario, has these values set at $(0.40, 0.51, 0.69)$ which ensures that the system is only loaded up to about 70% capacity, but for the most part is operating at between 40% and 50% capacity. Finally we define a low load scenario which uses the values $(0.07, 0.15, 0.25)$.

A. Soft Real-Time Scenarios

In the following experiments the penalty e_i for a job expiring is assumed to be zero. This means that the heuristics need only consider how to maximize utility accrual, and not about ensuring that certain jobs be scheduled. We begin by focusing on the high load scenario and calculating the value of each heuristic for 100 different 5 task problem instances as a percentage of value-optimal.

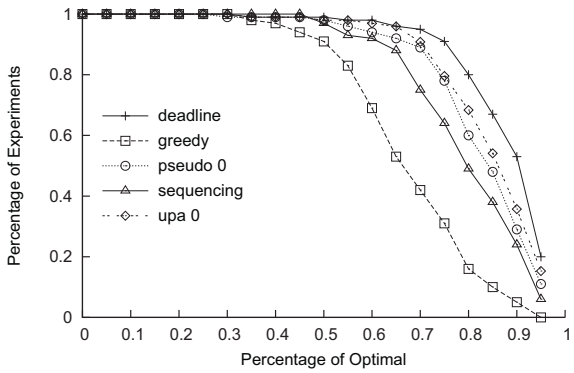
Figure 1 shows the results of our experiments. Each graph shows what percentage of the problem instances scheduled by each heuristic achieved a given percentage of optimal. Figure 1(a) shows that all the heuristics achieved at least 30% of optimal on all problem instances. In contrast, less than 20% of the problem instances scheduled using the greedy heuristic achieved at least 80% of optimal. This graph shows that for soft real-time cases with high load and downward step utility functions, the deadline heuristic performs best.

However, as can be seen in Figures 1(b) and 1(c), the deadline heuristic does not perform as well when the time utility functions are linear drop or target sensitive. Instead UPA 0 performs best, followed closely by Pseudo 0. However, as was discussed in Section IV-E, this marginal improvement in quality between Pseudo 0 and UPA 0 comes at the cost of a large jump in complexity. This particular value of α was chosen because of the experiments shown in Figure 1(d), which show that by a large margin a value of $\alpha = 0$ outperforms any other value of Pseudo for this particular case. Although the graph shown is only for linear drop utility functions, the results for downward step and target sensitive were nearly indistinguishable from it. For soft real-time scenarios the value of Pseudo α seems maximized by $\alpha = 0$, falls quickly and levels off for values not near 1 or 0, and then deteriorates rapidly again near 1.

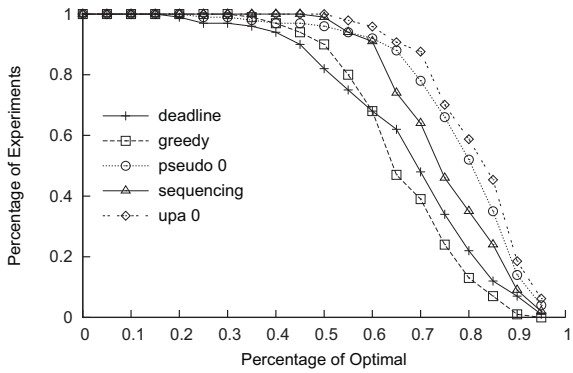
In all cases, the greedy heuristic performs relatively poorly, despite greedily maximizing utility density. This is most likely because the utility density is not strictly related to τ_i, and thus a job might have a higher immediate utility density, but still might not be the most urgent job. The sequencing heuristic also performs relatively poorly despite being much more computationally expensive than either UPA 0 or Pseudo 0. This is in some sense surprising for two reasons: (1) the scheduling decision made at every point is optimal if we assume that the schedule must be work conserving and that no more jobs arrive until the ready queue empties; and (2) UPA 0 uses a variation of sequencing to achieve its modest gains over Pseudo 0. Further investigation of this issue remains open as future work.

B. Hard Real-Time Scenarios

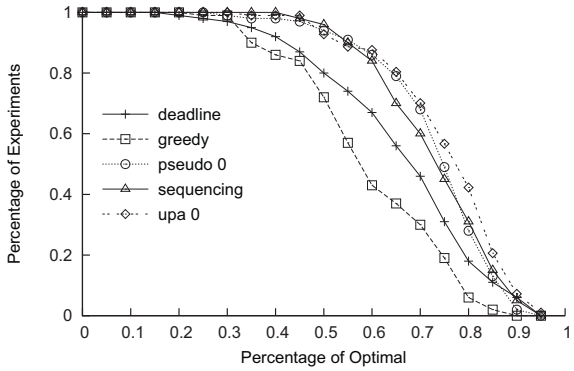
Unlike the soft real-time scenarios presented in Section V-A, in hard real-time systems a task expiring may incur severe penalties. Whereas in the soft real-time case, the only risk is



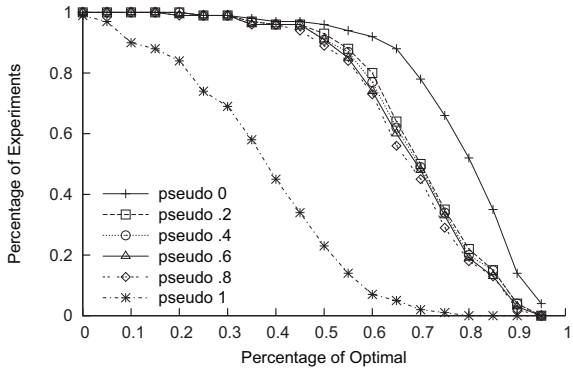
(a) Comparison of heuristics for downward step utility functions.



(b) Comparison of heuristics for linear drop utility functions.



(c) Comparison of heuristics for target sensitive utility functions.



(d) Effect of α on Pseudo α for linear drop utility functions.

Fig. 1. Comparison of heuristic policy performance for a soft real-time task set with five tasks under heavy load.

the potential loss of utility from not scheduling the job, while in the hard real-time case the penalty associated with the job, e_i, may be very large. For the experiments presented in this section one of the five tasks is assumed to be a real-time task, with e_i chosen uniformly at random from the range $(50, 150]$. For the other tasks in the system $e_i = 0$.

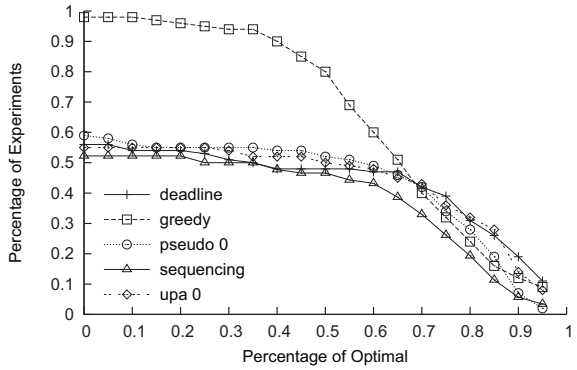
Figure 2 shows the results of these experiments. The first difference compared to the experiments in Section V-A is that some of the scheduling policies now achieve overall negative value: that is, they accrue more penalty in the long term than they achieve utility. Figure 2(c) shows that almost 10% of the problem instances scheduled by the greedy heuristic have negative value. Pseudo 0, in contrast, cannot gain positive value in 40% of the cases. Deadline, sequencing and UPA only have positive value in 50% of the cases.

The differences in performance by time utility type are also more muted in the hard real-time domain. However, Figure 2(a) shows that (as before) the deadline heuristic performs best when scheduling problem instances with downward step utility functions, but as Figures 2(b) and 2(c) show it does not do nearly as well in problem instances with linear drop or target sensitive utility functions. In Figure 2(c) Pseudo 0 is noticeably worse relative to other scheduling heuristics when scheduling jobs whose utility functions are target sensitive, whereas with other utility curves it performs approximately as well as deadline, Pseudo 0, sequencing, and UPA 0.

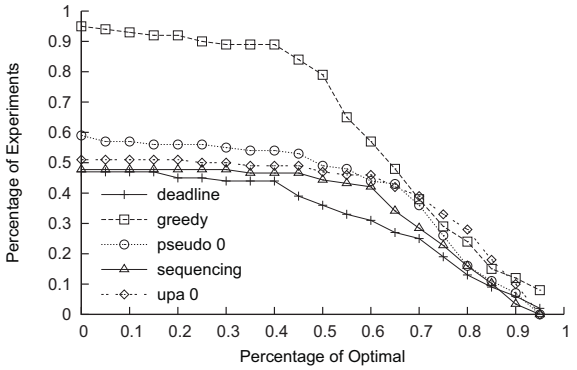
Although the greedy heuristic has the smallest number of problem instances with negative value it is important to remember that, as mentioned in Section IV-C, calculating greedy is more expensive in the hard real-time case because the immediate reward of a scheduling decision is no longer independent of other jobs in the ready queue. However, it is unclear why the sequencing heuristic, despite similar considerations, degrades sharply. It is possible that because sequencing more effeciently utilizes the resource, hard real-time jobs are more likely to arrive when the resource is occupied and thus expire before completion. Although we report results for Pseudo 0 and UPA 0, Figure 2(d) shows that for hard-real time problem instances, it is not clear that there is a single best parameter. In the problem instance shown here, the best value for α is 1, while our other experiments show strong evidence that this is the worst value for α in soft real-time problem instances. It is worth noting that although Pseudo 1 maximizes expected value, the value is still negative. Further investigation of these issues with the sequencing, Pseudo α, and UPA α heuristics remain open as future work.

C. Load Scenarios

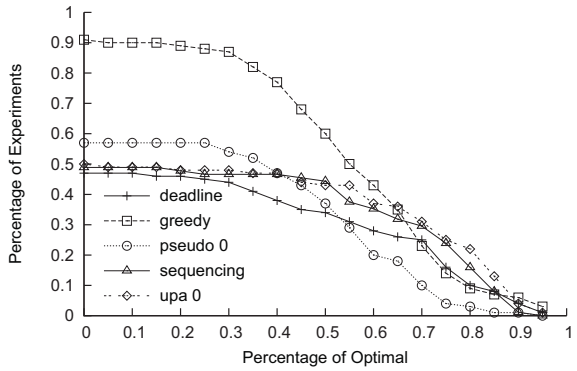
Figure 3 shows the effect of different loads on the quality of the scheduling heuristics. Figures 3(a), 3(c) and 3(e) show



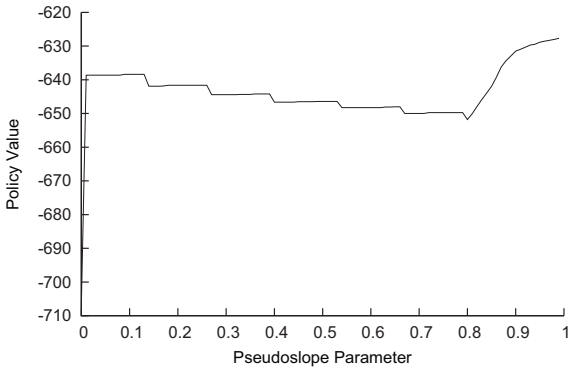
(a) Comparison of heuristics for downward step utility functions.



(b) Comparison of heuristics for linear drop utility functions.



(c) Comparison of heuristics for target sensitive utility functions.



(d) Effect of α on Pseudo α for a single problem instance with target sensitive utility functions.

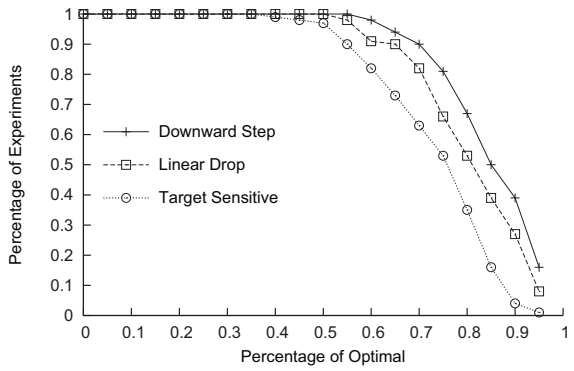
Fig. 2. Comparison of heuristic policy performance for a hard real-time task set with five tasks under heavy load.

soft-real time scenarios for different loads on Pseudo 0. As seen in Figure 3(a) Pseudo 0 incurred only minor differences in the quality of the schedules produced in the high load case, regardless of what time utility function class is being scheduled. While Pseudo 0 does noticeably better when scheduling jobs with downward step utility functions in high load cases when scheduling jobs with linear drop utility functions, these differences become even smaller in the medium load scenario shown in Figure 3(c). However, even as that happens the gap between problem instances with these utility functions and the target sensitive utility functions becomes more extreme. This trend continues in the low load scenario, shown in Figure 3(e). In this scenario Pseudo 0 achieves almost identical performance to a value-optimal scheduling algorithm when the time utility functions are either downward step or linear drop. However, problem instances where jobs have target sensitive utility functions are scheduled comparatively poorly.

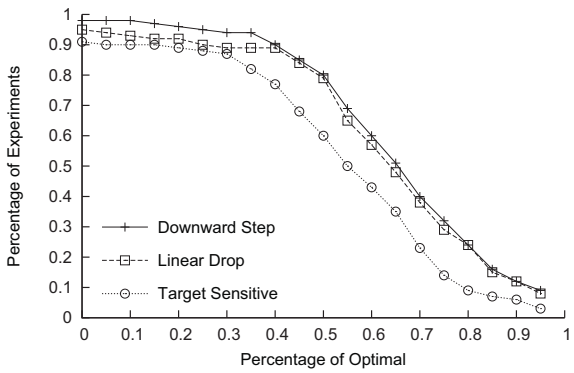
The poor performance of the Pseudo 0 heuristic in the medium and (especially) low load scenarios may be explained by two factors. First, with less resource contention a value-optimal policy has more degrees of freedom to optimize performance, and therefore heuristic policies like Pseudo 0 achieve a lower percentage of value-optimal. In essence there is more potential utility to be gained, and even a heuristic that achieves the same absolute value in different load scenarios would achieve a lower percentage of value-optimal in low load. Second, work conserving heuristics like Pseudo 0 in low load scenarios are more likely to schedule jobs early when there is little contention for the resource, even though that might result in lower overall expected utility. Figures 3(b), 3(d) and 3(f) show the effects of load in the hard real-time scenario for the greedy heuristic. The same trends that were seen in the soft real-time case are visible here as well. Once again for all but target sensitive utility functions, heuristics did best in the low load scenario, in which almost all problem instances achieved 90% of value-optimal. As in the soft real-time case, low loads made scheduling heuristics perform worse on target sensitive tasks, which was most likely caused by similar phenomenan to those described in the soft real-time case.

D. Other Time Utility Function Effects

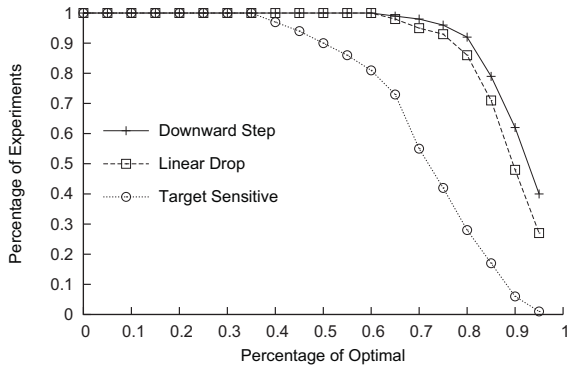
As was shown in Sections V-A and V-B, the shape of the time utility function can affect the quality of a scheduling heuristic. Because the shape of a time utility function can be arbitrary, an interesting question is whether particular families of curves are particularly difficult to schedule. To examine this effect we consider additional classes of time utility functions. Unlike the downward step, linear drop and target sensitive curves, these curves are not inspired by particular tasks in real-time or cyber-physical systems but rather by their potential



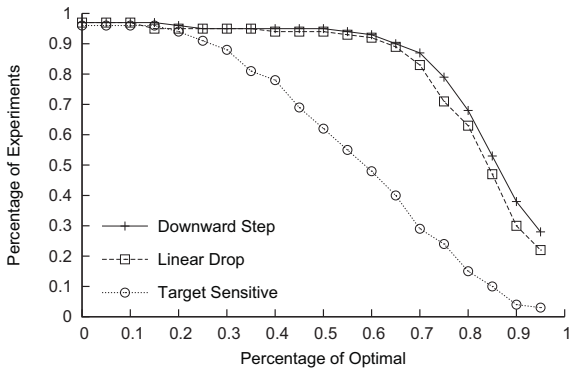
(a) Pseudo 0 in high load soft real-time scenarios.



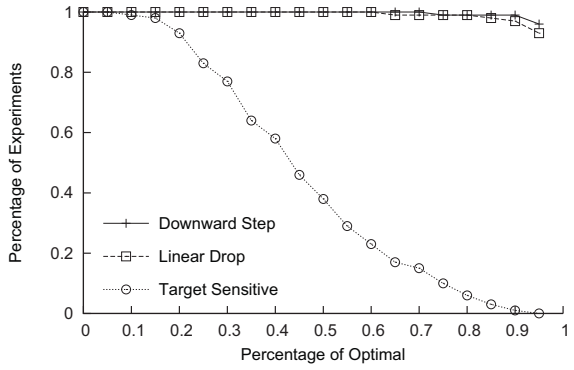
(b) Greedy in high load hard real-time scenarios.



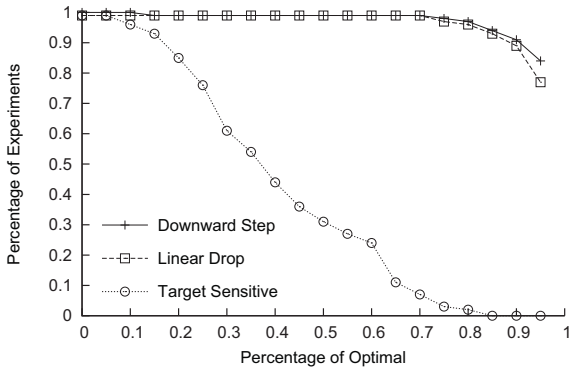
(c) Pseudo 0 in medium load soft real-time scenarios.



(d) Greedy in medium load hard real-time scenarios.



(e) Pseudo 0 in low load soft real-time scenarios.



(f) Greedy in low load hard real-time scenarios.

Fig. 3. Evaluation of selected heuristics for soft and hard real-time cases, for different time utility functions in low, medium and high load scenarios.

to reinforce or thwart assumptions pertaining to different heuristics. These curves are shown in Figure 4. The first is the *downward step function* utility curve, where utility falls off in a series of n flat discrete steps:

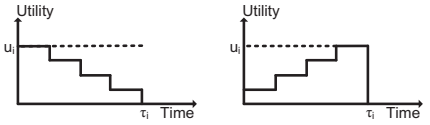
$$U_i(t) = \begin{cases} \frac{u_i}{n}\lceil\frac{nt}{\tau_i}\rceil & : \quad t < \tau_i \\ 0 & : \quad t \geq \tau_i \end{cases} \tag{8}$$

The *upward step function* utility curve is similar, but utility rises as the task approaches its termination time, and then falls off to zero afterward:

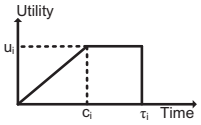
$$U_i(t) = \begin{cases} \frac{u_i}{n}(n - \lceil\frac{nt}{\tau_i}\rceil + 1) & : \quad t < \tau_i \\ 0 & : \quad t \geq \tau_i \end{cases} \tag{9}$$

The *rise linear* utility curve is a variation of the target sensitive utility curve, where utility rises up to a critical point c_i and then remains flat, but has an abrupt deadline:

$$U_i(t) = \begin{cases} t\frac{u_i}{c_i} & : \quad t < c_i \\ u_i & : \quad c_i \leq t < \tau_i \\ 0 & : \quad t \geq \tau_i \end{cases} \tag{10}$$



(a) Downward Step Function (b) Upward Step Function



(c) Rise Linear

Fig. 4. Possible time utility functions.

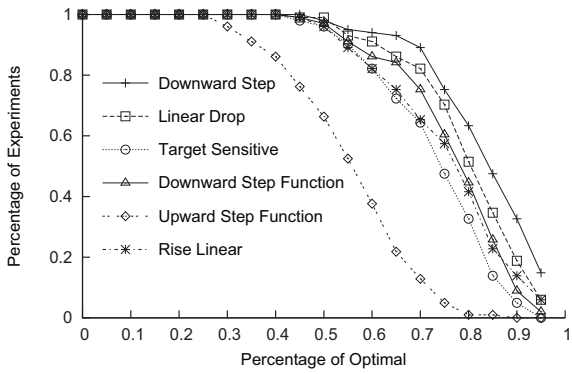


Fig. 5. Effects of time-utility function class on Pseudo 0.

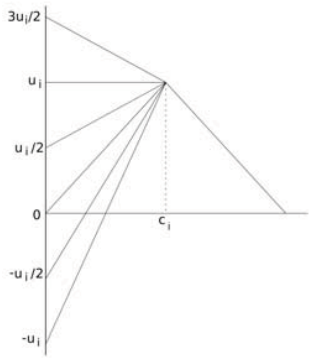


Fig. 6. Linear drop utility function, with different slope intercepts.

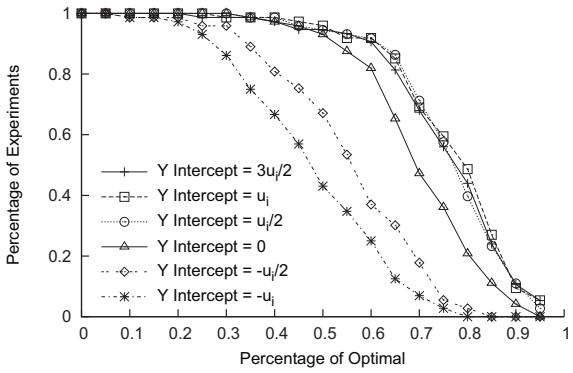


Fig. 7. Effects of initial time utility function slope on Pseudo 0.

To evaluate these effects we ran Pseudo 0 in the soft real-time high load scenario with problem instances created with all six classes of utility curves and 5 tasks. The results of running this experiment on 100 problem instances for each time utility function class are shown in Figure 5. Very little differentiates most classes of time utility functions, at least in the heavy load case. The notable exception to this is the upward step function. This likely occurs because Pseudo 0 tries to approximate the curve at any particular moment as a linearly decreasing function, and the upward step function behaves in a way contrary to this simplified utility model.

A final observation about the classes of time utility functions we consider in this paper is that target sensitive and linear drop only differ in the slope of the line before the critical point. By changing the slope of the utility curve before the critical point we get the class of utility functions as shown in Figure 6, to which both belong. Curves with Y intercept = 0 are target sensitive utility curves, and curves with Y intercept = u_i are linear drop curves. At any point where $U_i(t) < 0$ we assume instead that $U_i(t) = 0$. The effect of these utility curves on Pseudo 0 is shown in Figure 7. The problem instances shown in this experiment are soft real-time 5 task sets with high load. These experiments show that as the utility curve becomes more peaked, Pseudo 0 performs worse compared to value-optimal.

VI. Conclusions

In this paper we have examined scalable heuristics for utility aware scheduling of tasks with stochastically distributed non-preemptive execution intervals. In previous work we presented techniques for deriving value-optimal utility aware schedules [1], but computation costs for calculating the policy, and storage costs for online use of the policy, necessitated the exploration of the scalable solutions described here. Our evaluation framework allows us to compare various heuristics to a value-optimal policy in order to quantify their performance exactly despite stochastic task behavior.

From the experimental evidence presented in Section V the heuristic that provided the best compromise between scheduling complexity and performance for soft real-time systems was Pseudo 0, which is a generalization of UPA [2]. There were two notable exceptions, however. First, if the time utility functions of the tasks being scheduled were downward step utility functions, the deadline heuristic was superior to Pseudo 0. Second, in cases with low load and tasks with target sensitive utility functions, no general case scheduler achieved a significant percentage of the value-optimal objective.

For hard real-time task sets the only acceptable scheduler was the greedy heuristic, albeit with two caveats. First, the value of the policy is not guaranteed to be positive. Second, the scheduling overhead is proportional to the worst case exe-

	High Load	Medium Load	Low Load
Downward Step	Deadline	Deadline	Deadline
Linear Drop	Pseudo 0	Pseudo 0	Pseudo 0
Target Sensitive	Psuedo 0	Pseudo 0	None

(a) Soft Real-Time Guidelines

	High Load	Medium Load	Low Load
Downward Step	Greedy	Greedy	Greedy
Linear Drop	Greedy	Greedy	Greedy
Target Sensitive	Greedy	Greedy	None

(b) Hard Real-Time Guidelines

Fig. 8. Guidelines for soft and hard real-time scenarios.

cution time among jobs, which could make it computationally too expensive for online use. As in the soft real-time case, no general case scheduler performed well given low load and tasks with target sensitive utility functions. These guidelines are summarized in Figure 8.

This work thus reveals several open research issues in utility aware scheduling. First is the identification of a general case, scalable, utility aware scheduler for hard real-time systems with non-preemptable tasks with stochastic duration. One approach may be to address the scalability problems associated with the value-optimal scheduler's computation and/or storage cost. Second is the quantification and/or generalization of specialized utility aware schedulers in cases where general schedulers perform poorly may be worthwhile. For instance, the Gravitational Task Model [5] is ideally suited to low load and tasks with target sensitive utility functions and non-preemptive execution intervals, but not if the tasks have stochastic durations. We also plan to expand our evaluations of the heuristics' performance to include different mixtures of TUFs and different degrees of control over tasks completion times.

VII. Acknowledgements

This research has been supported in part by NSF grants CNS-0716764 (Cybertrust) and CCF-0448562 (CAREER). We also wish to thank the anonymous reviewers for their suggestions for this paper and for future work.

References

[1] T. Tidwell, R. Glaubius, C. D. Gill, and W. D. Smart, "Optimizing expected time utility in cyber-physical systems schedulers," in *RTSS'10: Proceedings of the 2010 Real-Time Systems Symposium*. Washington DC, USA: IEEE Computer Society, 2010.

[2] J. Wang and B. Ravindran, "Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 1, pp. 119–133, 2004.

[3] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley, "An adaptive, distributed airborn tracking system," in *IEEE Workshop on Parrallel and Distributed Real-Time systems*, 1999, pp. 353–362.

[4] D. Isovic, G. Fohler, and L. F. M. Steffens, "Timing constraints of mpeg-2 decoding for high quality video: Misconceptions and realistic assumptions," in *Proceeding of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, 2003.

[5] R. Guerra and G. Fohler, "A gravitational task model with arbitrary anchor points for target sensitive real-time applications," *Real-Time Ssytems*, vol. 43, no. 1, pp. 93–115, 2009.

[6] A. Anta and P. Tabuada, "On the benefits of relaxing the periodicity assumption for networked control systems over can," in *Proceedings of the 30th IEEE Real-Time Systems Symposium*, 2009.

[7] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha, "Coscheduling of cpu and i/o transactions in cots-based embedded systems," in *Proceedings of the 2008 Real-Time Systems Symposium*, 2008, pp. 221–231.

[8] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *Proceedings of the 1998 Real-Time Systems Symposium*. Washington DC, USA: IEEE Computer Society, 1998, pp. 123–132.

[9] G. Buttazzo and E. Bini, "Optimal dimensioning of a constant bandwidth server," in *Proceedings of the 27th IEEE Real-Time Systems Symposium*, 2006.

[10] N. Manica, L. Abeni, and L. Palopoli, "Reservation-based interrupt scheduling," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Application Symposium*. Stockholm, Sweden: IEEE Computer Society, 2010.

[11] P. K. Saraswat, P. Pop, and J. Madsen, "Task mapping and bandwidth reservation for mixed hard/soft fault-tolerant embedded systems," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Application Symposium*. Stockholm, Sweden: IEEE Computer Society, 2010.

[12] S. Manolache, P. Eles, and Z. Peng, "Schedulability analysis of applications with stochastic task execution times," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 4, pp. 706–735, 2004.

[13] A. F. Mills and J. H. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Application Symposium*. Stockholm, Sweden: IEEE Computer Society, 2010.

[14] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Inverted autonomous helicopter flight via reinforcement learning," in *International Symposium on Experimental Robotics*, 2004.

[15] W. D. Smart and L. P. Kaelbling, "Practical reinforcement learing in continuous spaces," in *Proceedings of the 17th International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 903–910.

[16] N. Kohl and P. Stone, "Machine learning for fast quadrupedal locomotion," in *Proceedings of the 19th National Conference on Artificial Intelligence*, 2004, pp. 611–616.

[17] R. Glaubius, T. Tidwell, W. D. Smart, and C. Gill, "Scheduling design and verification for open soft real-time systems," in *RTSS'08: Proceedings of the 2008 Real-Time Systems Symposium*. Washington DC, USA: IEEE Computer Society, 2008, pp. 505–514.

[18] R. Glaubius, T. Tidwell, B. Sidoti, D. Pilla, J. Meden, C. Gill, and W. D. Smart, "Scalable scheduling policy design for open soft real-time systems," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Application Symposium*. Stockholm, Sweden: IEEE Computer Society, 2010.

[19] R. Glaubius, "Scheduling policy design using stochastic dynamic programming," Ph.D. dissertation, Washington University in St. Louis, 2010.

[20] R. K. Clark, "Scheduling dependent real-time activities," Ph.D. dissertation, Carnegie Mellon University, 1990.

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

[22] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time systems," in *Proceedings of the 1985 Real-Time Systems Symposium (RTSS 1985)*, 1985, pp. 112–122.

[23] C. D. Locke, "Best-effort decision-making for real-time scheduling," Ph.D. dissertation, Carnegie Mellon University, 1986.

[24] P. Li, "Utility accrual real-time scheduling: Models and algorithms," Ph.D. dissertation, Virginia Tech, 2004.

[25] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 2005.

[26] C. L. Lui and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.