# Optimizing Expected Time Utility in Cyber-Physical Systems Schedulers

Terry Tidwell, Robert Glaubius, Christopher D. Gill and William D. Smart
Department of Computer Science and Engineering
Washington University in St. Louis
Email: {ttidwell,rlg1,cdgill,wds}@cse.wustl.edu

***Abstract*—Classical scheduling abstractions such as deadlines and priorities do not readily capture the complex timing semantics found in many real-time cyber-physical systems. Time utility functions provide a necessarily richer description of timing semantics, but designing utility-aware scheduling policies using them is an open research problem. In particular, scheduling design that optimizes expected utility accrual is needed for real-time cyber-physical domains.**

**In this paper we design scheduling policies that optimize expected utility accrual for cyber-physical systems with periodic, non-preemptable tasks that run with stochastic duration. These policies are derived by solving a Markov Decision Process formulation of the scheduling problem. We use this formulation to demonstrate that our technique improves on existing heuristic utility accrual scheduling policies.**

## I. Introduction

Computing systems increasingly interact with the physical world through sensing, actuation, or both. Timing constraints imposed by these interactions are of paramount concern for the safe and correct operation of these cyber-physical systems.

Because size, weight, power consumption or other physical restrictions may strongly influence their deployment and operation, such cyber-physical systems may be resource constrained and different jobs may contend for their shared common resources. When that happens, a scheduling policy must arbitrate access to shared resources while satisfying system timing and other constraints.

Real-time systems traditionally model timing constraints as deadlines. In *hard real-time systems* [1] any deadline miss may be considered equivalent to total system failure, or at least to the loss of any value from the corresponding job. Such a system is not feasibly schedulable unless all jobs meet all their deadlines. In contrast, *soft real-time systems* do not assume deadline misses are catastrophic. This implies some flexibility in how missed deadlines are handled. In some soft real-time systems, a deadline miss may mean that the job simply should be aborted. In others a late job should be completed with as little overrun as possible. However, none of these cases cover the full range of possible timing semantics that may be inherent to cyber-physical systems.

Time utility functions (TUFs) are a powerful abstraction for expressing more general timing constraints [2], [3], which characterize the *utility* of completing particular jobs as a function of time and thus capture more complex semantics than simple deadlines. To satisfy temporal constraints in a system whose semantics are described by time utility functions, a scheduling policy must maximize system-wide utility accrual.

However, jobs in cyber-physical systems have other features that complicate the scheduling problem. Specifically, jobs in many cyber-physical systems may be non-preemptable and may have stochastic duration. For example, jobs may involve actuation, such that preemption of a job may require restoring the state of a physical apparatus such as a robot arm. In such cases the cost of preemption may be unreasonably high. Therefore scheduling algorithms for cyber-physical systems must be able to consider non-preemptable jobs. Interaction with the physical environment also may lead to unpredictable job behavior, most notably in terms of job durations. A scheduling policy should anticipate this variability not only by considering the worst case execution time (as is often done in deadline-based scheduling) but also the probabilistic distribution of job durations. This is especially important when the goal is to maximize utility accrual, which depends on the timing of job completion.

Scheduling problems with these concerns may arise in a variety of cyber-physical systems as well as in traditional real-time systems. In mobile robotics, jobs competing for use of a robotic arm must be scheduled for efficient alternation. In CPU scheduling, quality of service (QoS) for an application may be specified as a time utility function, and the jobs to be scheduled may have long critical sections were preemption is not allowed. Finally, in critical real-time systems access to the bus by COTS peripherals may have to be scheduled in order to guarantee real-time performance [4].

In this paper we extend our previous work on Markov Decision Process (MDP) based scheduling policy design [5]–[7] to generate *utility-aware* scheduling policies appropriate for real-time cyber-physical systems with non-preemptable jobs that run with stochastic duration. This extension enables us to derive scheduling policies that optimize expected utility accrual for this important class of systems. It also improves on current state of the art utility accrual schedulers, which rely on heuristics and do not take advantage of predictable future job releases such as jobs released under a periodic task model.

This paper is organized as follows. In Section II we provide a brief survey of related work. In Section III we describe

This research has been supported in part by NSF grants CNS-0716764 (Cybertrust) and CCF-0448562 (CAREER).

1052-8725/10 $26.00 © 2010 IEEE
DOI 10.1109/RTSS.2010.28

our system and task model. In Section IV we formulate the scheduling problem as a Markov Decision Process and describe our solution approach. In Section V we examine the performance of scheduling policies for different classes of time utility functions in our system model. We offer conclusions in Section VI and propose directions for further investigation.

## II. Related Work

Time utility functions have been used to design and develop airborne tracking systems [8] to describe the value of track-association computations as a function of time. Time utility functions can also be used in control systems to specify the sensitivity of the control loop to inter-job jitter [9]. Deploying each job at a precise point in time gains maximal utility while deploying the job early or late can result in control instabilities [10].

Utility accrual scheduling primarily has been restricted to heuristics based on maximizing instantaneous potential utility density, which is the expected utility of running a job normalized by its expected duration [2].

The Generic Benefit Scheduler (GBS) [11] schedules tasks under resource contention using the potential utility density heuristic without assigning deadlines. If there is no resource contention, the proposed scheduling policy simply greedily schedules jobs according to the highest potential utility density.

Locke's Best Effort Scheduling Algorithm (LBESA) [12] schedules jobs with stochastic durations and non-convex time utility functions using a variation of Earliest Deadline First (EDF) [13], where jobs with the lowest potential utility density are dropped from the schedule if the system becomes overloaded. This technique requires an assignment of job deadlines along their time utility curves; optimal selection of those deadlines is itself an open problem.

Other research on utility accrual scheduling has crucially relied on restricting the shapes of the time utility curves to a single class of functions. The Dependent Activity Scheduling Algorithm (DASA) [14] assumes time utility functions are non-increasing downward step functions. The Utility Accrual Packet Scheduling Algorithm (UPA) [15], which extends an algorithm presented by Chen and Muhlethaler [16], assumes time utility functions can be approximated using a strictly linearly decreasing function. Gravitational task models [9] assume that the shapes of the time utility functions are symmetric and unimodal. In addition it is assumed that utility is gained when the non-preemptable job is scheduled, which is equivalent to assuming deterministic job durations with utility gained on job completion.

No existing utility accrual scheduling approach anticipates future job arrivals. Therefore, existing techniques are suboptimal for systems in which future arrivals can be accurately predicted, such as those encountered under a periodic task model [13].

MDPs have been used to model sequential decision problems including applications in cyber-physical domains such as helicopter control [17], [18] and mobile robotics [19], [20]. Our previous work [5]–[7] formulated MDPs to design scheduling policies in soft real-time environments with always-available jobs, but did so only for simple utilization-share-based semantics. In this work we extend such use of MDPs to design new classes of *utility-aware* scheduling policies for periodic tasks with stochastic duration.

Several other attempts have been made to address the difficulties that arise from non-preemptive and stochastic tasks in real-time systems. Statistical Rate Monotonic Scheduling (SRMS) [21] extends the classical Rate Monotonic Scheduling (RMS) [13] algorithm to deal with periodic tasks with stochastic duration. Constant Bandwidth Servers (CBS) [22] allow resource reservation in real-time systems where tasks have stochastic duration. Manolache, et al. [23], estimate deadline miss rates for non-preemptive tasks with stochastic duration. These works use classical scheduling abstractions such as priority and deadlines, rather than time utility functions, and are thus not appropriate for systems with the more complex timing semantics considered by our approach.

## III. System Model

We consider a system of non-preemptable jobs with stochastic durations and timing semantics formulated as time utility functions (TUFs). More formally, in our system model $n$ periodic tasks $(T_i)_{i=1}^{n}$ require mutually exclusive use of a single common resource. Each task $T_i$ consists of an infinite sequence of non-preemptable jobs, where $J_{i,j}$ denotes the $j$th job in $T_i$. A new job of $T_i$ is added to the ready queue every $p_i$ time units, where $p_i$ is the period of task $T_i$. Each job of a task has stochastic duration distributed according to probability mass function $D_i$, where $D_i(t)$ is the probability of the job's duration being exactly $t$. Each task has an associated time utility function $U_i(t)$ that maps the time elapsed since the release time $r_{i,j}$ of job $J_{i,j}$ to a real number representing the utility of completing that job at time $r_{i,j} + t$.

At each invocation the scheduler can choose to run any available job $J_{i,j}$ in the ready queue. Doing so results in the job occupying the scheduled resource stochastically for some number of quanta distributed according to $D_i$. The scheduler may instead choose to keep the resource idle for the next quantum. A *value-optimal* scheduling policy under this system model chooses scheduling actions (dispatching an available job on the scheduled resource or keeping the resource idle for a quantum) so as to maximize expected long term system utility accrual. Note that schedules produced under this model need not be work conserving (scheduling some job if any job is available) since a work conserving schedule may not be value-optimal given arbitrarily shaped time utility functions. For example, unimodal TUF $U_i$ may peak too late relative to job $J_{i,j}$ if the job were to be run at its earliest opportunity.

To make this MDP-based scheduling policy design approach feasible we place additional constraints on the task duration distribution, $D_i$, and the time utility function $U_i$. We assume that each task has a worst-case execution time $w_i$ that serves as an upper bound on the support of $D_i$. We also assume that each task has a termination time $tm_i$ after which $U_i(t)$

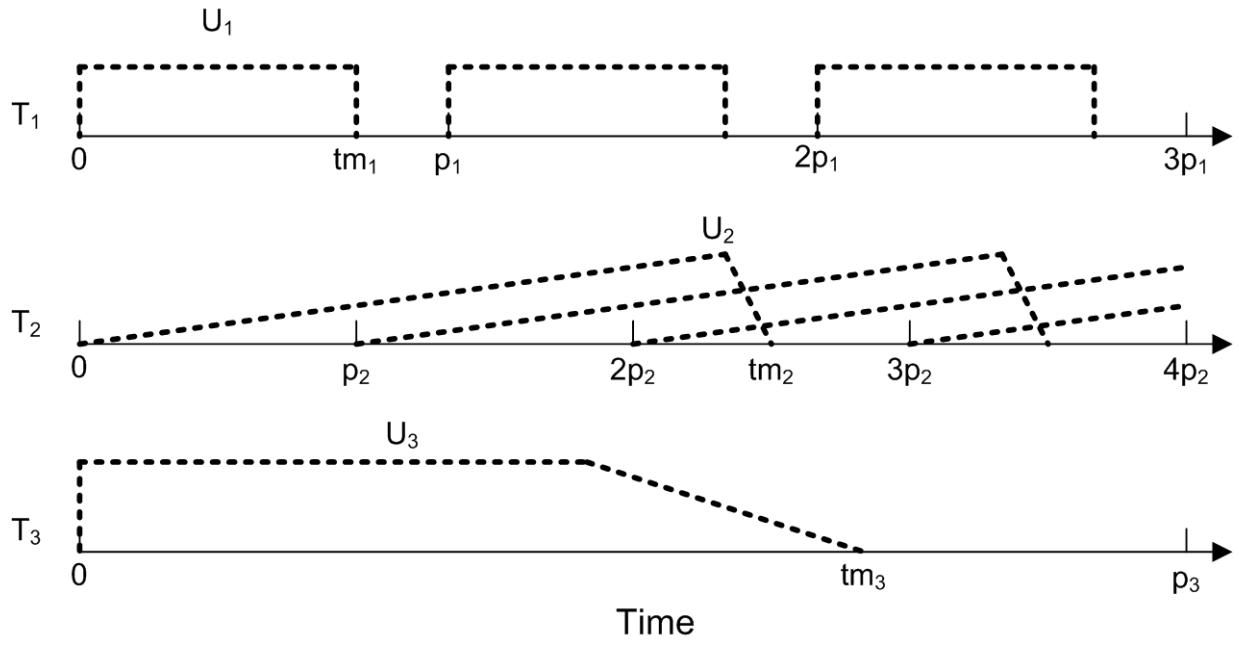



Fig. 1. Example 3 task system. Periods $p_i$, time utility curves $U_i$, and termination times $tm_i$ are shown for each task.

becomes zero. At time $r_{i,j} + tm_i$ job $J_{i,j}$ is removed from the ready queue, if it hasn't yet been chosen to be run. When this happens a penalty term $c_i$ is subtracted from the system's total accrued utility. We discuss these requirements in greater detail in Section IV based on the illustrative example task set shown in Figure 1.

## IV. Solution Approach

In previous work [5]–[7] we presented methods for formulating scheduling problems with non-preemptable jobs with stochastic duration as MDPs. Using existing techniques from the operations research literature [24] we were able to derive policies for a particular class of scheduling problems. That work focused on scheduling always-available non-preemptable jobs with stochastic durations to adhere to a desired resource share. This was achieved by penalizing the system in proportion to its deviation from the desired share target.

In this work we extend those techniques to design *utility-aware* rather than *share-aware* scheduling policies. In addition, we extend our previous work from an always-available job model to a periodic task model.

### A. Markov Decision Processes (MDPs)

An MDP is a five-tuple $(\mathcal{X}, \mathcal{A}, P, R, \gamma)$ consisting of a collection of states $\mathcal{X}$ and actions $\mathcal{A}$, a transition system $P$ that establishes the conditional probabilities $P(y|x, a)$ of transitioning from state $x$ to $y$ on action $a$, and a reward function $R$ that specifies the immediate utility of acting in each state. The reward function $R$ is defined over the domain of state-action-state tuples such that $R(x, a, y)$ is the immediate reward for taking action $a$ in state $x$ and ending up in state $y$. The discount factor $\gamma \in [0, 1)$ defines how potential future rewards are weighed against immediate rewards when evaluating the impact of taking action $a$ in a given state.

A policy $\pi$ for an MDP maps states in $\mathcal{X}$ to actions in $\mathcal{A}$. At each discrete decision epoch $k$ the agent observes the state of the MDP $x_k$, then selects an action $a_k = \pi(x_k)$. The MDP then transitions to state $x_{k+1}$ with probability $P(x_{k+1}|x_k, a_k)$ and the controller receives reward $r_k = R(x_k, a_k, x_{k+1})$. Given the discount factor $\gamma$, the *value* of a policy, written $V^\pi$, is the expected sum of long-term, discounted rewards obtained while following that policy:

$$V^\pi(x) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \middle| x_0 = x, a_k = \pi(x_k) \right\}. \quad (1)$$

Overloading notation, we let

$$R(x, a) = \sum_{y \in \mathcal{X}} P(y|x, a) R(x, a, y)$$

denote the expected reward when executing $a$ in $x$. Then we may equivalently define $V^\pi$ as the solution to the linear system

$$V^\pi(x) = R(x, \pi(x)) + \gamma \sum_{y \in \mathcal{X}} P(y|x, \pi(x)) V^\pi(y)$$

for each state $x$. When $|R(x, a)|$ is bounded for all actions in all states, the discount factor $\gamma$ prevents $V^\pi$ from diverging for any choice of policy, and can be interpreted as the prior probability that the system persists from one decision epoch to the next [25]. In practice this value is almost always set very close to 1 and in this work we set $\gamma$ to .99.

There are several algorithms, often based on dynamic programming techniques, for computing the optimal value for an MDP with finite state and action spaces. For the results

presented in Section V we employ modified policy iteration, but other standard techniques can be used, and we refer the reader to the book by Puterman [24] for a comprehensive survey. These solutions calculate the optimal value, $V^*(x)$, for every state $x \in \mathcal{X}$:

$$V^*(x) = \max_{a \in \mathcal{A}} \left\{ R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y) \right\}$$

The optimal action for every state can be found similarly:

$$\pi^*(x) = \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y) \right\}$$

The value-optimal policy is the policy that optimizes long term value, in contrast to immediate reward. Once computed, this policy is stored as a lookup table mapping states to actions.

### B. Utility-Aware Task Scheduling MDP

In our system model there are two salient features that determine the *scheduler state*: the system time, and the set of jobs available to run. We define our MDP over the set of scheduler states. The scheduler state is composed of a variable for tracking the elapsed time since the system was started, $\tau$, and indicator variables $q_i$ for each task that track whether there are any jobs of $T_i$ in the ready queue.

For the experiments presented in Section V we assume that $tm_i \leq p_i$. In this case $q_i$ is defined over $\{0,1\}$ and is 1 if there is a job of task $T_i$ in the ready queue and 0 otherwise. Because we require the termination time of the job to be less than or equal to the task period, we need only reason about one job per task at a time. For example, in Figure 1 tasks $T_1$ and $T_3$ have TUFs that satisfy this restriction, and so only a single job of each task may be in the ready queue at any time.

If we allow $tm_i > p_i$ but assume that jobs of the task must be run in order, $q_i$ is an integer which counts the number of jobs of $T_i$ that are in the ready queue. The values that $q_i$ can take are bounded by $\lceil tm_i/p_i \rceil$, the maximum number of jobs of $T_i$ that can be in the ready queue. Note that this MDP is a strict generalization of the case where $tm_i \leq p_i$. For example, in Figure 1 $2p_2 < tm_2 < 3p_2$. Consequently, at most three jobs of task $T_2$ may be in the ready queue.

If we allow $tm_i > p_i$ and also allow jobs of a task to be run out of order, $q_i$ becomes an array in $\{0,1\}^{\lceil tm_i/p_i \rceil}$ that tracks which of the $\lceil tm_i/p_i \rceil$ most recently released jobs of $T_i$ are in the ready queue. Note that this again is a strict generalization of the previous two cases.

An action $a_{i,j}$ in our MDP is the decision to dispatch job $J_{i,j}$ and is only valid in a scheduler state if $q_i$ indicates $J_{i,j}$ is in the ready queue. If we assume that $tm_i \leq p_i$ (or that jobs of $T_i$ must be run in order), only a single job of $T_i$ is ever eligible to be run. In this case we can simplify our set of actions to $a_i$, the action that runs that eligible job. In addition there is a special action $a_{idle}$, available in every state, which is the decision to advance time in the system by one quantum without scheduling any job.

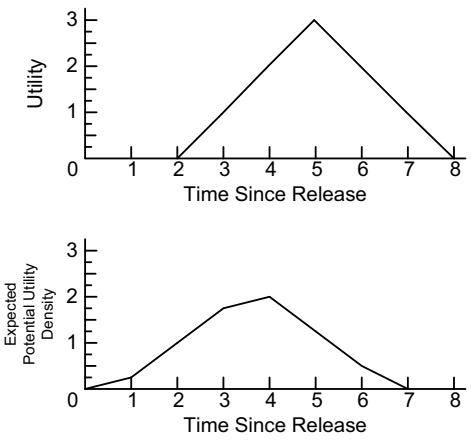



Fig. 2. Expected potential utility density.

With this mapping from scheduler states to MDP states in place, the transition function $P(y|x,a)$ is the probability of reaching scheduler state $y$ from $x$ when choosing action $a$. We discuss the formulation of the reward function for our utility aware task scheduling MDP next, in Section IV-C.

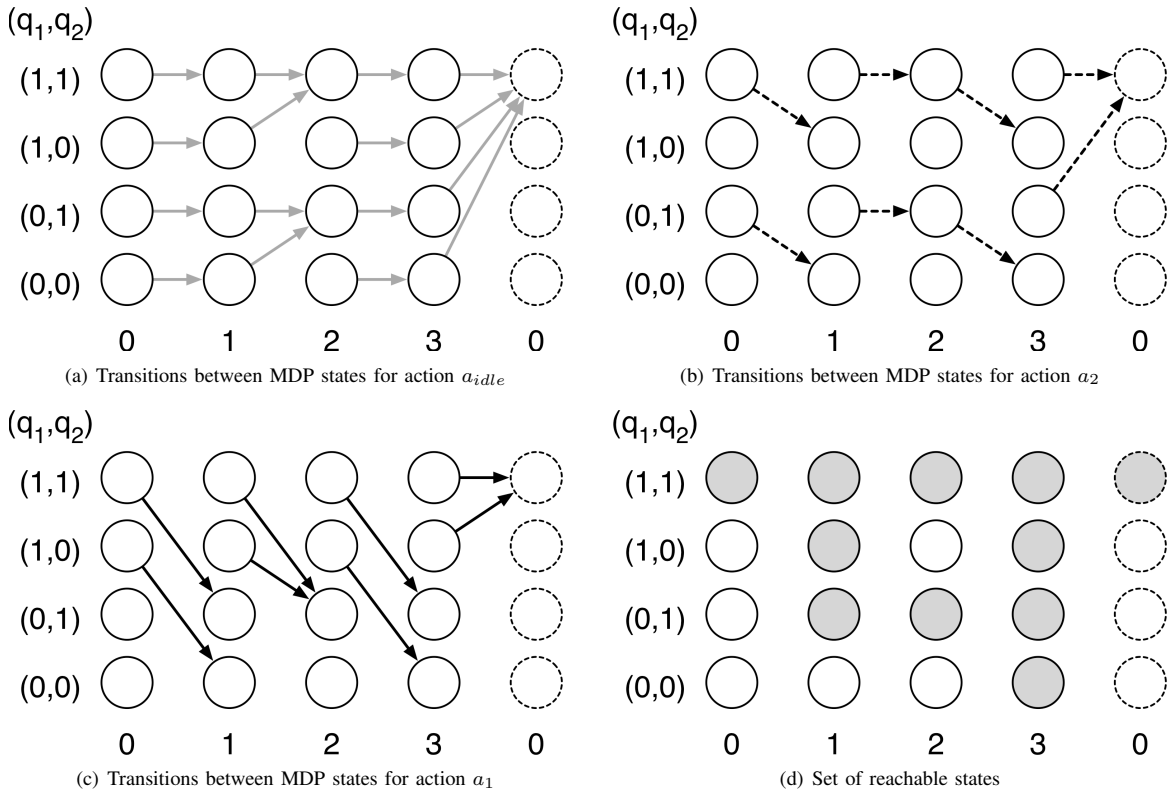
### C. Reward Function

The reward function is defined similarly to the transition function. The scheduler accrues no reward if the resource is idled, i.e., $R(x, a_{idle}, y)$ is always zero. Otherwise, $R(x, a_{i,j}, y)$ is the *utility density* of job $J_{i,j}$ of task $T_i$ that was just run. The utility density of a job $J_{i,j}$ completed at system time $\tau$ is defined as $U_i(\tau - r_{i,j})/(\tau - r_{i,j})$. Utility density is used as the immediate reward as opposed to $U_i(\tau - r_{i,j})$ in order to differentiate between jobs with different durations. It is then possible to define the expected potential utility density and thus the immediate reward for action $a_{i,j}$ in terms of $D_i$ and $U_i$, as a function of the current time $\tau$ and the release time of the job $r_{i,j}$ as shown in Equation 2.

$$R(x, a_{i,j}) = \sum_{d=1}^{w_i} \frac{D_i(d) U_i(d + \tau - r_{i,j})}{d} \tag{2}$$

An example calculation of expected potential utility density is shown in Figure 2. Task $T_i$ has TUF $U_i$ as shown in the upper graph. $D_i$ is defined on the range $[1,2]$ with $D_i(1) = 0.5$ and $D_i(2) = 0.5$. The bottom graph shows the expected utility density calculation shown in Equation 2 which is defined to be the immediate reward for scheduling the jobs of task $T_i$ at different times after that job's release.

Although the equation given in Equation 2 works for the general case in which any number of jobs of a task can be run in any order, if we assume $tm_i < p_i$ we can simplify $R(x, a_{i,j})$ to simply $R(x, a_i)$ because only one job of the task will be eligible to run. In this case, the time since the release of the job of task $T_i$ at time $\tau$ is $\tau$ mod $p_i$. Under this set of assumptions the expected immediate reward $R(x, a_i)$ is:




(a) Transitions between MDP states for action $a_{idle}$

(b) Transitions between MDP states for action $a_2$

(c) Transitions between MDP states for action $a_1$

(d) Set of reachable states

Fig. 3. Wrapped utility-aware task scheduling MDP for two tasks with periods $p_1 = 4$ and $p_2 = 2$, termination times equal to periods, and deterministic single quantum duration.

$$R(x, a_i) = \sum_{d=1}^{w_i} \frac{D_i(d)U_i(d + (\tau \bmod p_i))}{d} \tag{3}$$

Similarly, if we allow $tm_i > p_i$ but assume that jobs must be run in order the time since the release of the job of task $T_i$ scheduled by $a_i$ is $p_i(q_i - 1) + (\tau \bmod p_i)$ and the immediate reward is:

$$R(x, a_i) = \sum_{d=1}^{w_i} \frac{D_i(d)U_i(d + p_i(q_i - 1) + (\tau \bmod p_i))}{d} \tag{4}$$

At time $r_{i,j} + tm_i$ job $J_{i,j}$ is removed from the ready queue if it has not been run. We account for this by subtracting a cost term $C(x, a)$ from the immediate reward function:

$$C(x, a) = \sum_{i=1}^{n} e_i c_i \tag{5}$$

The term $e_i$ is the expected number of jobs of task $T_i$ that will expire given that action $a$ is chosen in state $x$, and the term $c_i$ is the penalty for the job of task $T_i$ expiring. For real-time tasks where deadline misses are catastrophic $c_i$ would be set to negative infinity. In this paper we assume that for all tasks $c_i = 0$ and that the only penalty incurred by the system is missing the chance to gain any utility for running the job.

### *D. Wrapped Utility-Aware Task Scheduling MDP*

Because $\tau$ is not bounded the resulting MDP has an infinite number of states. However, the hyperperiod $H$ of the tasks, defined as the least common multiple of the task periods, allows us to wrap the state space of our MDP into a finite set of exemplar states as follows. The intuition is similar to that for hyperperiod analysis in classical real-time scheduling approaches. Given two states $x$ and $y$ with identical $q_i$ for all tasks and times that map to the same point in the system hyperperiod, $\tau_x \bmod H = \tau_y \bmod H$, the two states will have the same relative distribution over successor states and rewards. This means that the value-optimal policy will be the same at both states. Thus it suffices to consider only the finite subset of states where $\tau < H$. Anytime the MDP would transition to a state where $\tau \geq H$ it instead transitions to the otherwise identical state with system time $\tau \bmod H$.

An example of a wrapped utility-aware task scheduling MDP is shown in Figure 3. In this example there are two tasks $T_1$ and $T_2$ with deterministic quantum durations ($D_1(1) = D_2(1) = 1$), and termination times equal to their period ($tm_1 = p_1 = 4$, and $tm_2 = p_2 = 2$). The states of the MDP are projected so that states with the same $\tau$ are arranged in columns, and states with the same tuple of indicator variables $(q_1, q_2)$ are arranged in rows. Figures 3(a), 3(b), and 3(c) show the transitions between states for taking action $a_{idle}$, $a_1$ and $a_2$ respectively. The states where $\tau = 0$ are duplicated on

the right side of each figure to illustrate our state wrapping over the hyperperiod, which for the example shown is 4. Figure 3(d) shows the set of reachable states from the state where $q_1 = q_2 = 1$ and $\tau = 0$, which is the system's initial state.

Such state wrapping bounds the values that each of the state variables can take, and consequently bounds the size of the MDP. If we assume $tm_i \leq p_i$, an upper bound for the number of states in the scheduling MDP is

$$2^n H. \tag{6}$$

If instead we allow $tm_i > p_i$ but assume jobs of a task must be run in order, an upper bound for the number of states in the scheduling MDP is

$$H \prod_{i=0}^{n} \lceil tm_i/p_i \rceil. \tag{7}$$

If we allow $tm_i > p_i$ but let jobs of a task run in arbitrary order, an upper bound for the number of states is

$$H 2^{\sum_{i=0}^{n} \lceil tm_i/p_i \rceil}. \tag{8}$$

However, the size of the state space is sensitive not only to the number of tasks and their constraints but also to the size of the hyperperiod. As such, the solution approach presented in this paper is especially appropriate for systems with harmonic tasks since in this case $H = \max(p_1, \ldots, p_n)$, which reduces the size of the state space that must be considered.

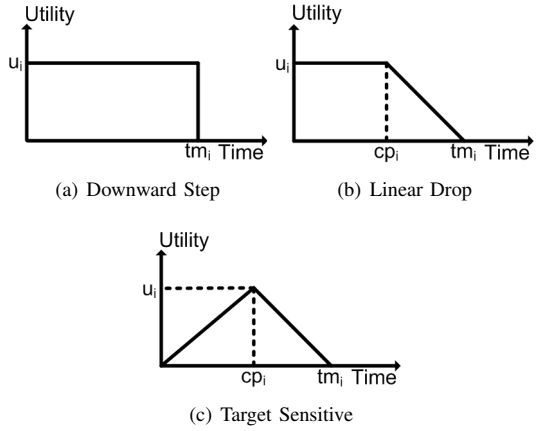
In any case, the wrapped MDP has finitely many states. Consequently it can be solved using existing techniques from the operations research literature [24]. As described in Section IV-A, the resulting policy for this scheduling MDP optimizes the value function given in Equation 1, which is the discounted sum of expected immediate rewards. Because these immediate rewards are defined to be the expected utility density of the scheduling decision, the policy produced optimizes utility accrual, and is value-optimal in the sense that no policy exists that gains more value in expectation. In practical terms this means that the produced scheduling policy is the best possible utility accrual scheduling policy for the modeled system.

The produced policy is stored in a lookup table for use at run-time. The cost of hashing the scheduler state is proportional to its size, resulting in $O(a)$ time overhead, where $a$ is the number of actions available to the scheduler. For the assumptions made in this paper this is equivalent to $O(n)$, where $n$ is the number of tasks.

## V. Experimental Evaluation

Although the solution outlined in Section IV can be applied to arbitrary time utility functions, for the purposes of evaluation we concentrate on three particular classes of representative time utility functions, of the forms shown in Figure 4.

These three classes of time utility functions are both able to represent a wide range of timing constraints and suitable for evaluating the approach presented in this paper. The three




(a) Downward Step (b) Linear Drop

(c) Target Sensitive

Fig. 4. Representative time utility functions.

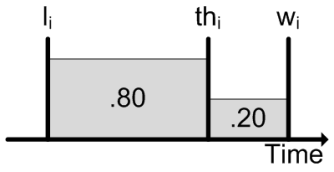



Fig. 5. Task duration distributions used in experiments.

classes are parameterized linear versions of curves found in real systems [3]. The first curve, shown in Figure 4(a), is representative of time utility functions for hard real-time jobs, whose utility remains constant up until a deadline after which the job provides no utility[1], e.g., jobs in real-time avionic or automotive systems. In our experiments this family of curves is parameterized by two variables: $tm_i$, a termination time after which the expected utility of the tasks falls to zero, and $u_i$ the upper bound on the utility curve, which for the experiments presented here was chosen uniformly at random from the range [2,32].

The second curve, shown in Figure 4(b), is representative of time utility functions for soft real-time jobs whose utility degrades with time but suffers no sharp drop, e.g., calculation of distant waypoints in on-board flight planning [26]. In our experiments this family of curves is parameterized by three variables: $tm_i$, a termination time after which the expected utility of the tasks falls to zero; $u_i$ the upper bound on the utility curve (chosen uniformly at random from the range [2,32]); and $cp_i$ the critical point at which the expected utility of the job starts to fall linearly toward zero (chosen uniformly at random from the range [0,$tm_i$]).

The third curve, shown in Figure 4(c), is representative of time utility functions for target sensitive jobs - jobs sensitive to jitter but less sensitive to delays, e.g., in control systems where inter-job jitter can cause instabilities and failure [27], or in media-streaming applications in which buffering is not

[1]The case where a deadline miss implies system failure can be modeled via a similar step-function TUF but with an associated infinite penalty if the job misses its deadline.

possible since inter-frame jitter can cause degradation of quality [10]. In our experiments this family of curves is parameterized by three variables: $tm_i$, a termination time after which the expected utility of the tasks falls to zero; $u_i$ the upper bound on the utility curve (chosen uniformly at random from the range [2,32]); and $cp_i$ the critical point before which the expected utility rises linearly toward $u_i$ and after which the expected utility of the job starts to fall linearly toward zero (chosen uniformly at random from the range [0,$tm_i$]).

Task sets were created by randomly generating duration distributions and periods. Periods were randomly chosen factors of 2400 in the range [100, 2400]. Choosing periods for tasks this way ensured that the hyperperiod of the task set was at most 2400.

The duration distributions are balanced as shown in Figure 5, and are parameterized by three values: a lower bound $l_i$ and upper bound $w_i$ on the duration, and a threshold point $th_i$ such that:

$$\sum_{t=l_i}^{th_i} D_i(t) = 0.80$$

$$\sum_{t=th_i+1}^{w_i} D_i(t) = 0.20$$

In addition, these points were chosen to meet the following constraints:

$$w_i/p_i > th_i/p_i \geq l_i/p_i \geq 0.05$$

$$th_i/p_i \geq 0.10$$

$$\sum_{i=1}^{n} l_i/p_i = 0.70$$

$$\sum_{i=1}^{n} th_i/p_i = 0.90$$

$$\sum_{i=1}^{n} w_i/p_i = 1.20$$

The values of the duration distribution are uniform in the ranges $[l_i, th_i]$ and $[th_i + 1, u_i]$. Intuitively these constraints ensure that the task load on the system is normally between 0.70 and 0.90, with occasional transient overloads up to 1.20.

To evaluate the performance of our approach we consider a greedy policy $\pi_g$ that chooses the action that yields the best immediate reward in expectation as defined in Equation 3.

$$\pi_g(x) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \{R(x, a)\}$$

As noted in Section IV-C taking the action with the highest expected immediate reward from scheduler state $x$ corresponds to scheduling the job with the highest expected potential utility density. As discussed in [11] this is equivalent to the scheduling decision made by the Generic Benefit Scheduler (GBS) in the special case where there is no resource contention between jobs. We use this policy as a baseline for comparing how much improvement is made by solving for the value-optimal scheduling policy which optimizes long term value, over using a greedy heuristic approach. This greedy heuristic has $O(n)$ time overhead for online scheduling, where $n$ is the number of tasks, but has smaller memory overhead than our scheduling approach, as no lookup table is needed to store the schedule.

The quality of $\pi_g$, and $\pi^*$, the greedy and value-optimal policies respectively, can be compared directly using our MDP model by evaluating the value of each policy at the initial state, denoted $V^{\pi_g}(0)$ and $V^*(0)$ respectively. Because our approach produces scheduling policies that are optimal with respect to long term value, any other scheduling policy can be evaluated by what percentage of $V^*(0)$ it achieves.

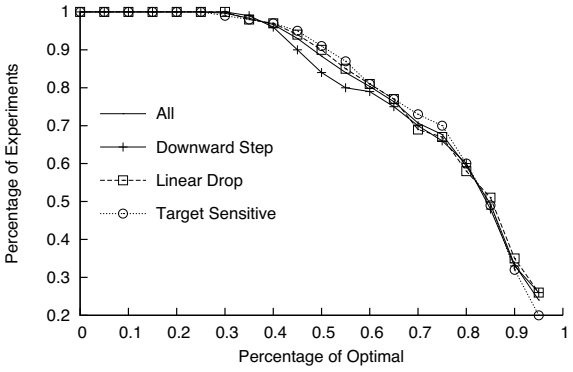
Figures 6(a), 6(b), 6(c) and 6(d) show results of comparing $V^{\pi_g}(0)$ and $V^*(0)$ for a set of randomly generated task scheduling problems each with 2, 3, 4 or 5 tasks respectively. For each task set size, 270 problem instances were generated: 90 for each time utility function of the three classes shown in Figure 4. Each graph shows the percentage of random problem instances where $V^{\pi_g}(0)$ was at least some percentage of $V^*(0)$. For example, in the 3 task problem instances shown in Figure 6(b), almost all the problem instances resulted in the $\pi_g$ having $V^{\pi_g}(0)$ at least 25% of $V^*(0)$. In only about 20% of problem instances was $V^{\pi_g}(0)$ at least 90% of $V^*(0)$. There was very little variation for the respective quality of the exact and heuristic scheduling approaches for the different classes of time utility functions. However, target sensitive time utility functions resulted in better performance overall for $\pi^*$ compared to $\pi_g$. One reason for this is that the greedy scheduling policy is work conserving, and thus cannot let the scheduled resource become idle while there is an available job. The value-optimal scheduling policy, on the other hand, can delay the start of the job in order to maximize the expected potential utility density.

Figure 7 shows the effect of the number of tasks on the relative scheduling policy quality for the heuristic and value-optimal approaches. As Figure 7 illustrates, as the number of tasks increases the quality of the heuristic scheduling policy deteriorates relative to the value-optimal policy.

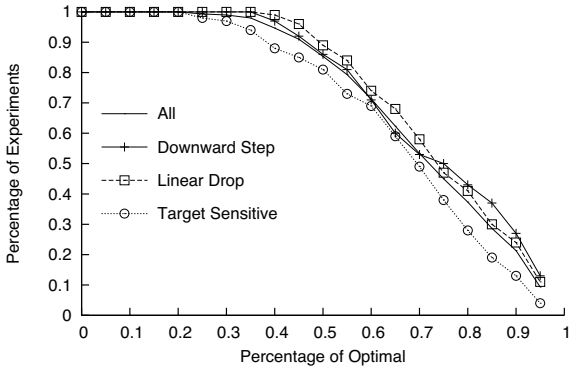
These experiments show that existing heuristic approaches for utility accrual scheduling can perform poorly when tasks are non-preemptable and have stochastic duration. This is especially true as the number of tasks grows and if the jobs are target sensitive. In contrast, the policies produced by our approach maximize utility accrual in the face of these concerns.

## VI. Conclusions

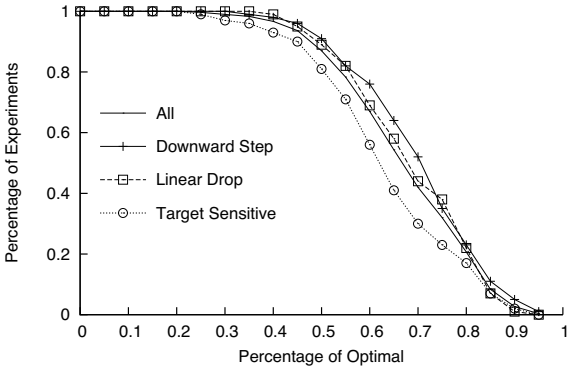
In this paper we have introduced novel MDP based utility-aware scheduling policy design techniques for systems with periodic tasks and jobs that are non-preemptive and run with stochastic duration. This approach yields scheduling policies



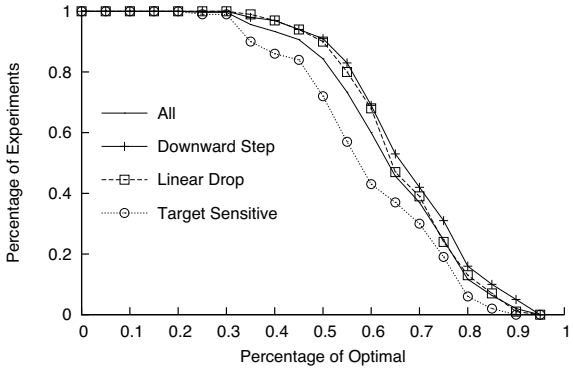

(a) Comparison of policy performance for 2 task scheduling problems.




(b) Comparison of policy for 3 task scheduling problems.




(c) Comparison of policy performance for 4 task scheduling problems.




(d) Comparison of policy performance for 5 task scheduling problems.

Fig. 6. Comparison of greedy policy performance to MDP-based value-optimal policies for scheduling problems with different classes of time utility functions and varying number of tasks.

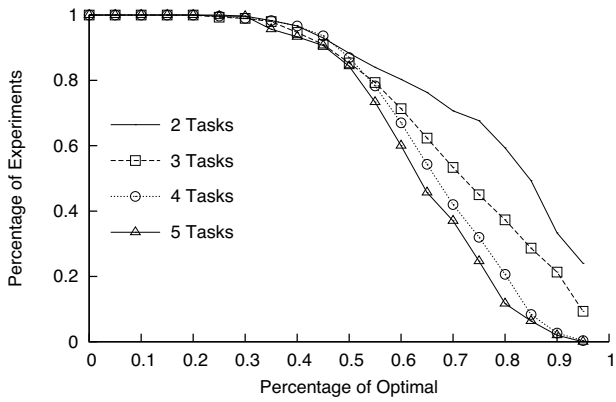



Fig. 7. Comparison of greedy policy performance to MDP-based value-optimal policies for scheduling problem instances with increasing number of tasks.

which optimize utility accrual and are appropriate for use in a variety of cyber-physical systems. These techniques are especially appropriate for systems with harmonic tasks, which reduces the size of the state space for the scheduling MDP.

Our utility-aware task scheduling MDP formulation also provides a framework for comparing the effectiveness of heuristic utility-aware scheduling policies. Our experiments demonstrate the superiority of MDP based scheduling policies over state of the art heuristic utility accrual schedulers for several representative classes of time utility functions. Our approach is especially well suited for system configurations where the performance of heuristic policies is weakest: as the number of tasks increases and when time utility functions for the tasks are target sensitive. A more extensive characterization of when existing heuristics perform poorly is an area that merits further study.

Improving the scalability of the MDP based techniques presented here also merits further investigation. We plan to explore several approaches to that problem as future work. One approach is to develop parameterized policies that approximate the behavior of the value-optimal policies found by solving the utility-aware task scheduling MDP. A second approach is to look at simplified utility-aware task scheduling MDPs which approximate the structure of the original while being less sensitive to exponential growth in the number of states. A third approach is to consider compositional and hierarchical approaches, which use a polynomial number of exponentially smaller MDPs.

## REFERENCES


[1] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2005.

[2] 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.

[3] B. Ravindran, E. D. Jensen, and P. Li, "On recent advances in time/utility functions real-time scheduling and resource management," in *Proceeding of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing 2005 (ISORC 2005)*, 2005, pp. 55–60.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

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

[15] 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.

[16] 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.

[17] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry, "Autonomous helicopter flight via reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 16. MIT Press, 2004, pp. 799–806.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

[25] L. P. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[26] C. D. Gill, D. C. Schmidt, and R. K. Cytron, "Multi-paradigm scheduling for distributed real-time embedded computing," *IEEE Proceedings, Special Issue of Modeling and Design of Embedded Systems*, vol. 91, no. 1, pp. 183–197, 2003.

[27] P. Marti, G. Fohler, K. Ramamritham, and J. M. Fuertes, "Jitter compensation in real-time control systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, London, UK, 2001.