Scheduling Design and Verification for Open Soft Real-time Systems *

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

Abstract

Open soft real-time systems, such as mobile robots, experience unpredictable interactions with their environments and yet must respond both adaptively and with reasonable temporal predictability. New scheduling approaches are needed to address the demands of such systems, in which many of the assumptions made by traditional real-time scheduling theory do not hold. In previous work we established foundations for a scheduling policy design and verification approach for open soft real-time systems, that can use different decision models, e.g., a Markov Decision Process (MDP), to capture the nuances of their scheduling semantics.

However, several important refinements to the preliminary techniques developed in that work are needed to make the approach applicable in practice. This paper makes three main contributions to the state of the art in scheduling open soft real-time systems: (1) it defines a novel representation of the scheduling state space that is both more compact and more expressive than the model defined in our previous work; (2) it exploits regular structure of that representation to allow efficient verification of properties involving both discrete and continuous system state variables under specific scheduling policies; and (3) it removes the unnecessary use of a time horizon in our previous approach, thus allowing the more precise specification and enforcement of a wider range of scheduling policies for open soft real-time systems.

1 Introduction

Open soft real-time systems, such as mobile robots, must respond adaptively to varying operating conditions. In many situations, such as when a mobile robot approaches a physical obstacle or a branch in its possible navigation path, decisions must be made and enacted within specific timing constraints (e.g., before hitting the obstacle or passing a waypoint at which a different path should have been taken). In many cases fail-safe actions, such as having the robot temporarily pause its motion, are available in the event of an anticipated or detected timing failure in these systems. However, taking a fail-safe action constitutes sub-optimal operation of the system (e.g., the robot will take longer to reach its objective if it repeatedly pauses its motion), and the design of such systems therefore must strive to meet these timing constraints to the extent possible.

Three main challenges must be addressed in the design of scheduling policies for these systems: (1) since many open soft real-time systems have power and storage constraints as well as timing constraints, their scheduling policies must use CPU and memory resources efficiently at run-time; (2) schedulability analysis, which for open soft real-time systems in general amounts to a verification problem due to the possible complexities of the scheduling state space for such systems, must be tractable; and (3) precise specification and enforcement of the scheduling policies in the context of real systems and their environments must be supported, so that system behavior is rigorously controlled in practice. How to ensure efficient, verifiable, and precise scheduling of the CPU and other resources in open soft real-time systems is thus an important and challenging research problem.

Section 2 summarizes related work on resource monitors and scheduling policy design. In recent work [1] we developed a method for scheduling policy design that can be tailored to specified workloads, using an approach based on a Markov Decision Process (MDP), which considers the effects of scheduling threads in different interleavings, up to a specified time horizon. This in turn allowed model checking over finite execution histories to discover the set of possible system traces, and to verify specific system properties under the designed scheduling policy. In this paper we assume the same system model, which is described in Section 3, but overcome the following limitations of our previous approach: (1) the number of states in the MDP was exponential in the value of the time horizon, so that scheduling over longer intervals became prohibitively expensive in both design duration and run-time storage; (2) the decision procedure produced edge effects in the form of minor deviations from the specified policy near the time horizon; (3) our verification approach required coarse over-approximations to

*This research was supported in part by NSF grants CNS-0716764 and CCF-0448562.

achieve tractability, but still suffered from exponential complexity with respect to the history size; and (4) neither our scheduling policies nor our verification approach addressed continuous time semantics.

This paper presents several novel advances in the design and verification of scheduling policies for open soft real-time systems. First and foremost, it introduces a state wrapping technique, that as we discuss in Section 4 allows a scheduling policy to be encoded as an MDP with a greatly reduced number of states. It also eliminates the need for setting an artificial time horizon and thus eliminates the edge effects associated with our previous technique. Because of the compact representation of the MDP, it is easier to create policies derived from more densely sampled states, allowing better approximations of policies with continuous time semantics. As we discuss in in Section 5, verification of policy effects using the wrapped state space does not require pessimistic over-approximations, so that full exploration is much less expensive. Furthermore, the improved verification approach supports model checking of properties with continuous time semantics. In Section 6 we summarize our contributions, and describe planned future work.

2 Related Work

Resource Monitors: Separation kernels and other resource monitors can provide stringent enforcement of system resource allocation policies, but unfortunately existing approaches do so inflexibly, by segregating resources into discrete partitions [2, 3]. For example, the MILS kernel [3] partitions memory and CPU resources into separate virtual machines on which processes then execute, controlling not only access to resources, but also communication between processes running in different partitions. Through such strict separation, these approaches allow formal specification and verification [4] of resource allocation and use.

Scheduling Policy Design: Many thread scheduling policies have been developed to ensure feasibility of resource use in closed real-time systems [5]. These approaches typically assume that the number of tasks, and their invocation rates and execution times, are all well characterized. Approaches that allow even basic extensions such as asynchronous task arrival must depend on special services (e.g., admission control [6]) to maintain resource feasibility at run-time.

Hierarchical scheduling techniques [7, 8, 9, 10] offer greater flexibility in their ability to enforce scheduling policies adaptively at run-time, according to multi-faceted scheduling decision functions that are arranged hierarchically into a single system scheduling policy. However, there has been little prior work on verification of what guarantees can be made by such hierarchical scheduling policies. Furthermore, verification of scheduling policies that induce thread preemption and require reasoning about continuous time may encounter problems with decidability [11], so that special techniques that exploit knowledge about the structure of the specific scheduling problem [12, 13] may be needed before the techniques we are developing can be applied to systems with more nuanced execution semantics.

Dynamic programming has long been used for large-scale scheduling problems, such as those encountered in large machine shops [14]. A related technique, Reinforcement Learning (RL) [15] (often called Approximate Dynamic Programming), has been applied to several domains, including computer cluster management [16] and network configuration repair [17], and job scheduling [18]. An eventual goal of our work is to support scheduler design using techniques like RL for open soft real-time systems, but that topic is beyond the scope of this paper.

3 Background

We are developing new scheduling techniques that are flexible in the policies they can enforce, and in the properties that can be verified under those policies. In this section we summarize background material for the research contributions presented in Sections 4 and 5.

System Model: In previous work [1], we proposed an abstract system model in which: (1) multiple threads of execution require mutually exclusive use of a single common resource (i.e., a CPU); (2) whenever a thread is granted the resource, it occupies the resource for a finite and bounded subsequent duration; (3) the duration for which a thread occupies the resource may vary from run to run but overall obeys a known independent and bounded distribution over any reasonably large sample of runs of that thread; and (4) a scheduler repeatedly chooses which thread to run according to a given scheduling policy, dispatches that thread, and waits until the end of the duration during which the thread occupies the resource.

This system model establishes a foundation for the kinds of scheduling enforcement problems that can arise in open soft real-time systems built atop commonly used operating systems such as Linux or VxWorks. For example, within the Linux kernel, hard and soft interrupts may be threaded and placed under scheduler control [12], with different resulting durations of resource occupation for the different kinds of interrupts. As future work we plan to extend our approach to address issues such as preemption among *inter-dependent* intervals of execution, which this system model does not support.

We represent the state of the system in terms of the resource utilization of each thread of execution. To ensure that working within the system was tractable, in our previ-

ous work we introduced a fixed time horizon into the model, which enforces a bound on the number of possible utilization states. This assumption is quite restrictive, since we often do not know in advance the system's life span, or the intervals of system execution over which scheduling decisions must be evaluated.

In this paper we extend our previous work to eliminate the dependence on a time horizon by taking advantage of recurrence in the system model. This allows us instead to use a discounting scheme that can be thought of as a prior probability that a system will continue to execute from time step to time step. This allows us to derive scheduling policies that are optimal to arbitrary planning horizons.

Scheduling Decision Model: As in [1], our scheduling decision model consists of sequentially deciding to dispatch one of n threads whenever the CPU becomes available. The scheduler's objective is to maintain the relative resource utilization for each thread near some target utilization level, encoded in a vector $\mathbf{u}$. Threads release the CPU after some time; the execution time of thread a is non-deterministic with known distribution P_a. We make the simplifying assumption that run-times for subsequent executions of the same thread are independent, so that historical thread executions do not color future thread executions.

We represent this scheduling decision model as a Markov Decision Process (MDP) [19]. An MDP is a four-tuple (X, A, R, P), where X is a set of process states and A is a set of available actions. The transition function $P(y|x, a)$ describes the probability of entering state y after taking action a from state x. The reward function $R(x, a, y)$ indicates the immediate cost or benefit or moving from x to y on a. It is often convenient to consider the expected reward $R(x, a) = \sum_{y \in X} P(y|x, a)R(x, a, y)$.

A policy π recommends actions in each state of an MDP. Our objective is to discover a policy that maximizes the *value* obtained within the MDP. We focus on the discounted reward setting; given a discount factor $\gamma \in (0, 1]$, the value of a policy π at a state x is the discounted cumulative reward obtained by following policy π, defined as:

$$V^{\pi}(x) = E\left\{ \sum_{\tau=0}^{\infty} \gamma^{\tau} R(x_{\tau}, \pi(x_{\tau})) \middle| x_0 = x \right\} \quad (1)$$

This places greater emphasis on rewards that can be reached in fewer steps. In the model described in this paper, the infinite horizon model (i.e., $\gamma = 1$), results in $V^{\pi} = -\infty$ for any policy π, thus some discounting strategy is necessary in order to distinguish between policies. The optimal policy, π^*, maximizes Equation 1. Its value function V^{π^*}, or more compactly V^*, satisfies the system of Bellman equations

$$V^*(x) = \max_{a \in A} \{Q^*(x, a)\} \quad (2)$$

$$Q^*(x, a) = R(x, a) + \gamma \sum_{y \in X} P(y|x, a)V^*(y) \quad (3)$$

This optimality condition can be used to define the *value iteration* algorithm for approximating the optimal *value function* V^*. This algorithm computes the value of behaving optimally for τ steps. By convention, we begin with $V_0 = 0$, and compute V_{τ} recursively according to

$$V_{\tau}(x) = \max_{a \in A} \{Q_{\tau}(x, a)\}$$

$$Q_{\tau}(x, a) = R(x, a) + \gamma \sum_{i=1}^{n} P(y|x, a)V_{\tau-1}(y).$$

This algorithm converges asymptotically on V^* in weighted supremum norm $\| \cdot \|_{\mu}$,

$$\|V\|_{\mu} = \sup_{x \in X} |V(x)|\mu(x)^{-1},$$

provided that a positive weighting function $\mu(x)$ exists such that all of the iterates V_{τ} have $\|V_{\tau}\|_{\mu} < \infty$. This is satisfied by the weighting function $\mu = 1$ when there are finitely many states and the expected rewards are bounded.

In practice we use V_{τ} for some suitably large τ to approximate V^*. Our approximation π of the optimal policy π^* behaves greedily with respect to V_{τ}:

$$\pi(x) \in \operatorname*{argmax}_{a \in A} \{Q_{\tau+1}(x, a)\} \quad (4)$$

The closer V_{τ} is to V^*, the closer the corresponding policy is to the optimal policy [20]. When the set of policies is finite, the greedy policies corresponding to the value iterates generated by value iteration converge in finite time.

Utilization State Model: In our previous work [1], the scheduling decision state space consists solely of *utilization states* $\mathbf{x} \in \mathbb{Z}_+^n$, where $\mathbb{Z}_+$ is the non-negative integers. Component x_a of $\mathbf{x}$ gives the cumulative historical CPU utilization of thread a, so that the sum of components $\sum_{a=1}^{n} x_a$ is the total CPU usage over the entire system history. Since CPU usage is strictly non-negative, we can write this more compactly as $\|\mathbf{x}\|_1$, where $\| \cdot \|_1$ is the Manhattan distance.

Scheduling actions consist of the decision to run thread $a \in A = \{1, 2, \ldots, n\}$. We assume that the run-time distributions for each thread a, P_a, are known. We use these to define the state transition probabilities

$$P(\mathbf{y}|\mathbf{x}, a) = \begin{cases} P_a(t) & \mathbf{y} = \mathbf{x} + t\Delta_a \\ 0 & otherwise, \end{cases} \quad (5)$$

where $\Delta_a = (\delta_{a1}, \delta_{a2}, \ldots, \delta_{an})$, and δ_{ab} is the Kronecker delta. This means that the system can only move between states that differ only in terms of the utilization of thread a; the probability of this transition equals the probability $P_a(t)$ that thread a runs for $t = y_a - x_a$ time quanta.

We formulate rewards in this model based on the deviation from target utilization $\mathbf{u} \in \mathbb{R}_+^n$. We aim to provide thread a u_a percent of the CPU time, with the requirement

that the per-thread targets u_a satisfy $0 < u_a < 1$ and sum to one. The reward function $R(\mathbf{x}, a, \mathbf{y})$ depends on the distance of $\mathbf{y}$ from the projected best utilization at time $\|\mathbf{y}\|_1$, $\|\mathbf{y}\|_1\mathbf{u}$.

$$R(\mathbf{x}, a, \mathbf{y}) = r(\mathbf{y}) \quad (6)$$

$$r(\mathbf{x}) = -\|\mathbf{x} - \|\mathbf{x}\|_1 u_a\|_1 \quad (7)$$

Figure 1 illustrates this model for an example with two threads.

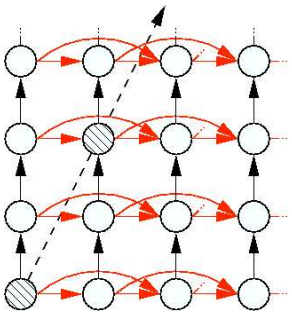

Figure 1: Utilization state space graph for a system with two threads. Each vertex is a utilization state; edges represent transitions with non-zero probability. Thread one is shown in red, thread two is shown in black. The target utilization ray is shown by the dashed arrow. States marked with a hashed pattern achieve target utilization.

If an MDP has a countable state space, and the expected rewards $\sup_{\mathbf{x},a} |R(\mathbf{x}, a)| < \infty$, then the value iteration algorithm is closed over a Banach space

$$\mathcal{V} = \{V : X \to \mathbb{R} | \sup_{\mathbf{x} \in X} V(\mathbf{x}) \leq R_{\max}/(1-\gamma)\}.$$

Existence of the optimal value function is guaranteed because the value iteration udpate is a contraction mapping with contraction parameter γ on this space.

The penalty function in Equation 7 cannot be bounded below, so that the result mentioned above does not hold; the value iteration algorithm can in general diverge. A standard approach is to show convergence in some weighted supremum norm

$$\|V\|_\mu = \sup_{\mathbf{x} \in X} |V(\mathbf{x})| \, \mu(x)^{-1}, \quad (8)$$

where $\inf_{\mathbf{x} \in X} \mu(\mathbf{x})$ is positive. Given such a norm, we instead consider value functions taken from the Banach space $\mathcal{V}_\mu = \{V : X \to \mathbb{R} | \|V\|_\mu < \infty\}$. Proving convergence in $\mathcal{V}_\mu$ of value iteration then reduces to finding μ so that $\mathcal{V}_\mu$ is closed under the value iteration update.

It is sufficient for convergence to show that there exists μ such that the following two conditions hold [19] for all states $\mathbf{x}$ and actions a:

$$\sup_{a \in A} |R(\mathbf{x}, a)| < C_1 \mu(\mathbf{x}) \quad (9)$$

$$\sum_{\mathbf{y} \in X} P(\mathbf{y}|\mathbf{x}, a)\mu(\mathbf{y}) \leq \mu(\mathbf{x}) + C_2 \quad (10)$$

where C_1 and C_2 are non-negative constants that hold for all states and actions.

In order to demonstrate convergence, we show that the above conditions are satisfied when $\mu(\mathbf{x}) = |r(\mathbf{x})| + \epsilon$ for arbitrary $\epsilon > 0$. First, Equation 9:

$$\begin{aligned}
|R(\mathbf{x}, a)| &= |\sum_{\mathbf{y} \in X} P(\mathbf{y}|\mathbf{x}, a) R(\mathbf{x}, a, \mathbf{y})| \\
&= \sum_{t=0}^{\infty} P_a(t) |r(\mathbf{x} + t\Delta_a)| \\
&= \sum_{t=0}^{\infty} P_a(t) \|\mathbf{x} + t\Delta_a - \|\mathbf{x} + t\Delta_a\|_1 \mathbf{u}\|_1 \\
&\leq \sum_{t=0}^{\infty} P_a(t) \left[\|\mathbf{x} - \|\mathbf{x}\|_1 \mathbf{u}\| + t\|\Delta_a - \mathbf{u}\|_1\right] \\
&= |r(\mathbf{x})| + |r(\Delta_a)| \sum_{t=0}^{\infty} t P_a(t) \\
&\leq \mu(\mathbf{x}) + |r(\Delta_a)| \sum_{t=0}^{\infty} t P_a(t) \leq C_1 w(\mathbf{x})
\end{aligned}$$

where

$$C_1 = 1 + \frac{\sup_{\mathbf{a} \in \mathbf{A}} \{|r(\Delta_a)| \sum_{t=0}^{\infty} t P_a(t)\}}{\inf_{\mathbf{y} \in X} w(\mathbf{y})}.$$

Since C_1 does not depend on $\mathbf{x}$, this is sufficient to show that the weighted maximum reward for each is bounded at each state. If the mean run times for each distribution $\sum_{t=0}^{\infty} t P_a(t)$ is finite, C_1 is finite, since $|r(\Delta_a)$ is constant for any action. The same argument can be used to show that Equation 10 is satisfied, since

$$\begin{aligned}
\sum_{t=0}^{\infty} P_a(t)\mu(\mathbf{x}) &= \sum_{t=0}^{\infty} P_a(t) \left[|r(\mathbf{x} + t\Delta_a)| + \epsilon\right] \\
&= |R(\mathbf{x}, a)| + \epsilon \\
&\leq w(\mathbf{x}) + \sup_{a \in A} \left\{ |r(\Delta_a)| \sum_{t=0}^{\infty} t P_a(t) \right\}.
\end{aligned}$$

Therefore, it follows that the optimal value function V^* is in $\mathcal{V}_\mu$, and value iteration can be shown to converge to V^* asymptotically.

In practice, we cannot work directly in the utilization state space due to its infinite size. In our previous work, we addressed this by truncating the utilization state space by

adding a termination time T. We restricted consideration to only those states with total utilization $\|\mathbf{x}\|_1 \leq T$. This does make convergence easy to establish, since the number of states is finite and expected rewards are bounded, but the policies obtained from this model suffered from minor deviations from specified policy due to boundary effects near the termination time.

A further problem with this approach is that the state space is large in both the number of threads n and the termination time. The number of states $|X|$ in this truncated model is $|X| = \sum_{i=0}^{T} \binom{n+i-1}{n-1}$, which grows exponentially in T and n. In Section 4 we describe a new approach that takes advantage of the similarity of the transition function to preserve optimality while eliminating the dependence on an artificially introduced termination time, by "wrapping" the state space in order to line up states with similar futures.

4 Wrapped State Model

The intuition behind the main contribution of this paper is to notice that the marked states in Figure 1 are highly similar: they have the same reward, and the relative distribution of future states is the same. We would expect the optimal policy to prefer the same action in these states. The key idea of our approach is to detect similar states and collapse them together to obtain a smaller, more tractable model.

We say that a system is *periodic* with period k if and only if k is a positive integer satisfying $\rho k\mathbf{u} \in \mathbb{Z}_+^n$ for all non-negative integers ρ, which means that the utilization ray $\lambda\mathbf{u}$, $\lambda \in \mathbb{R}_+$, passes through utilization states at regular intervals. In Figure 1, $\mathbf{u} = (1/3, 2/3)$, so $k\mathbf{u}$ is integer-valued for any $k > 0$ that is a multiple of three. We are particularly interested in the minimum period, which in this case is $k = 3$. Lemma 1 establishes the existence of periodic systems and provides a method for determining the minimum period for a broad class of utilization-based systems.

Lemma 1 *Suppose $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ such that for each a, $0 < u_a < 1$ is rational, and let p_a and q_a be the least terms satisfying $u_a = p_a/q_a$; i.e., $GCD(p_a, q_a) = 1$. Then the system is periodic with minimum period $k = LCM(q_1, q_2, \ldots, q_n)$.*

Proof: By definition of the least common multiple, for each index a there is a positive integer m_a such that $k = m_a q_a$. For any $\rho \in \mathbb{Z}_+$, $\rho k u_a = \rho k(p_a/q_a) = \rho m_a p_a \in \mathbb{Z}_+$, so the system is periodic with period k.

Suppose $l < k$ is a period. Then for any a, $lu_a = z_a \in \mathbb{Z}_+$. If for every a there is an m_a such that $l = m_a q_a$, then $l = k$, so there must be some index b such that for any $m_b \in \mathbb{Z}_+$, $l \neq m_b q_b$. Since $z_b = lu_b = l(p_b/q_b)$, $l = (z_b q_b)/p_b$. z_b/p_b is not an integer; nor is q_b/p_b since $GCD(q_b, p_b) = 1$. Thus, $(z_b q_b)/p_b = l$ is not an integer, so l cannot be a period. □

We use the period of a utilization state model to wrap the state space. One can visualize our wrapping method in two dimensions as first drawing the utilization states on a sheet with the state $(0, 0)$ in the bottom left corner, as in Figure 1. The sheet extends to infinity away from this corner. The wrapping proceedure consists of rolling up the sheet to form a tube so that the states $\rho k\mathbf{u}$ rest atop one another. We treat utilization states that rest atop one another as equivalent in this wrapped model.

The key idea behind the wrapped state model is that the distance from the initial utilization $(0, 0, \ldots, 0)$ is not all that important when choosing how to act. Historical utilization states matter only to the extent they determine the distribution over future utilization states, and that distribution only matters to the extent that it predicts the deviation from the utilization ray. By wrapping up the state space, we are enumerating unique possible deviations from target utilization while pruning out states that encode redundant deviation information. This results in a significant reduction in the number of MDP states; further, we can do this without loss of the optimal scheduling policy for the original utilization state model.

One interesting note is that we do not need to consider the thread run-time distributions when determining the state wrapping; it depends only on the period and target utilization vector. This is true under our assumption that the thread run-time distributions are independent of the utilization state. This allows us to collapse equivalent wrapped states without losing information about the distribution of future states.

We will show that the value function is periodic with period k, in the sense that the optimal value function $V^*(\mathbf{x}) = V^*(\mathbf{x} + \rho k\mathbf{u})$ for any $\rho \in \mathbb{Z}_+$. This relies on the fact that the penalty function is periodic:

$$\begin{aligned} r(\mathbf{x} + \rho k\mathbf{u}) &= -\|\mathbf{x} + \rho k\mathbf{u} - \|\mathbf{x} + \rho k\mathbf{u}\|_1\mathbf{u}\|_1 \\ &= -\|\mathbf{x} + \rho k\mathbf{u} - \|\mathbf{x}\|_1\mathbf{u} + \rho k\mathbf{u}\|_1 \\ &= -\|\mathbf{x} - \|\mathbf{x}\|_1\mathbf{u}\|_1 = r(\mathbf{x}) \end{aligned}$$

This holds because $\mathbf{u}$, $\mathbf{x}$, k, and ρ are non-negative and $\|\mathbf{u}\|_1 = 1$. As a consequence, the expected reward function is also periodic:

$$\begin{aligned} R(\mathbf{x} + \rho k\mathbf{u}, a) &= \sum_{t=0}^{\infty} P_a(t) r(\mathbf{x} + t\Delta_a + \rho k\mathbf{u}) \\ &= \sum_{t=0}^{\infty} P_a(t) r(\mathbf{x} + t\Delta_a) \\ &= R(\mathbf{x}, a) \end{aligned}$$

With these results, we can show that the optimal value function is also periodic.

Theorem 1 *For all states* $\mathbf{x}$ *and non-negative integers* ρ*,*

$$V^*(\mathbf{x}) = V^*(\mathbf{x} + \rho k\mathbf{u}) \tag{11}$$

Proof: We will inductively show that, for all $\mathbf{x}$, $V^*(\mathbf{x}) = V^*(\mathbf{x}+\rho k\mathbf{u})$ by showing that $V_\tau(\mathbf{x}) = V_\tau(\mathbf{x}+\rho k\mathbf{u})$ when $V_0 = 0$, in which case $V_0(\mathbf{x}) = V_0(\mathbf{x}+\rho k\mathbf{u}) = 0$. Suppose $V_\tau(\mathbf{x}) = V_\tau(\mathbf{x}+\rho k\mathbf{u})$. Then

$$\begin{aligned}
& Q_{\tau+1}(\mathbf{x}, a) \\
= \;& R(\mathbf{x}, a) + \gamma \sum_{t=0}^{\infty} P_a(t) V_\tau(\mathbf{x} + t\Delta_a) \\
= \;& R(\mathbf{x} + \rho k\mathbf{u}, a) + \gamma \sum_{t=0}^{\infty} P_a(t) V_\tau(\mathbf{x} + t\Delta_a + \rho k\mathbf{u}) \\
= \;& Q_{\tau+1}(\mathbf{x} + \rho k\mathbf{u}, a)
\end{aligned}$$

Therefore

$$\begin{aligned}
V_{\tau+1}(\mathbf{x}) &= \max_{a \in A}\{Q_{\tau+1}(\mathbf{x}, a)\} \\
&= \max_{a \in A}\{Q_{\tau+1}(\mathbf{x} + \rho k\mathbf{u}, a)\} \\
&= V_{\tau+1}(\mathbf{x} + \rho k\mathbf{u})
\end{aligned}$$

□

Since the Q-values for each action are the same at $\mathbf{x}$ and $\mathbf{x} + \rho k\mathbf{u}$ for every non-negative integer ρ, the set of optimal actions at periodic states is the same. This means that there exist optimal, periodic policies as well.

In particular, this result shows that if a set of states are colinear along the utilization ray translated to pass through any one of them, then it is necessary to store only a single value for all of those states. These states are equivalent with respect to the value function, so it is sufficient to construct a policy that makes decisions based solely on which equivalence class of states the current utilization state belongs to. We compute the value for the "smallest" member of equivalence classes of states $\{\mathbf{x} + \rho k\mathbf{u} | \rho \in \mathbb{Z}_+\}$ given $\mathbf{x}$ in $\mathbb{Z}_+^n$, which we can think of as wrapping up the state space in the direction $\mathbf{u}$. To describe this wrapping procedure formally, we specify a vector modulus operator $w : \mathbb{Z}_+^n \times \mathbb{Z}_+^n \to \mathbb{Z}_+^n$,

$$w(\mathbf{x}, \mathbf{v}) = \mathbf{x} - \rho\mathbf{v} \tag{12}$$

where

$$\rho = \max\{\rho' \in \mathbb{Z}_+ | \mathbf{x} - \rho'\mathbf{v} \in \mathbb{Z}_+^n\} \tag{13}$$

For compactness, we let $w(\mathbf{x})$ denote $w(\mathbf{x}, k\mathbf{u})$, where k is the minimum period for the system with n threads and target utilization $\mathbf{u}$. Using the above results, we know that every state $\mathbf{x}$ with $w(\mathbf{x}) = \mathbf{y}$ has the same value, so we only need to compute the value at these states. In Figure 1, the marked states are equivalent to each other and to all other utilization states along the utilization ray.

The transition function between wrapped states is defined by analogy to Equation 5:

$$P(w(\mathbf{y}) | w(\mathbf{x}), a) = \begin{cases} P_a(t) & w(\mathbf{y}) = w(\mathbf{x} + t\Delta_a) \\ 0 & otherwise \end{cases} \tag{14}$$

One concern could be that t may not be unique: that $w(\mathbf{y})$ equals both $w(\mathbf{x} + t_1\Delta_a)$ and $w(\mathbf{x} + t_2\Delta_a)$, with distinct values t_1 and t_2. Suppose that this is the case. Then

$$\begin{aligned}
& w(\mathbf{x} + t_1\Delta_a) = w(\mathbf{x} + t_2\Delta_a) \\
\equiv \;& \mathbf{x} + t_1\Delta_a - \rho_1 k\mathbf{u} = \mathbf{x} + t_2\Delta_a - \rho_2 k\mathbf{u} \\
\equiv \;& (t_1 - t_2)\Delta_a = (\rho_1 - \rho_2)k\mathbf{u}
\end{aligned}$$

If $\rho_1 = \rho_2$, then $t_1 = t_2$. If $\rho_1 - \rho_2 \neq 0$, $\mathbf{u} = \frac{t_1 - t_2}{k(\rho_1 - \rho_2)}\Delta_a$, a degenerate target utilization in which only thread a should be utilized. We are not concerned with such cases because the optimal scheduling policy is then trivial. Therefore, the transition model in Equation 14 is well-defined. Figure 2 illustrates this wrapping method on the example from Figure 1. This process basically takes the original collection of utilization states and deletes all of the utilization states that are componentwise no less than $k\mathbf{u}$, then reattaches transitions to deleted states to their equivalents among the wrapped states.

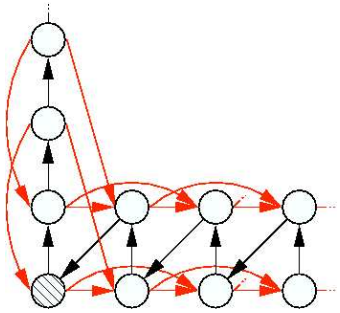

Figure 2: The wrapped state model corresponding to the utilization state model in Figure 1. Vertices represent equivalence classes and edges represent transitions for thread one (red) and thread two (black).

Following Equations 6 and 7, we define the reward $R(w(\mathbf{x}), a, w(\mathbf{y}))$ in terms of the penalty of entering $w(\mathbf{y})$,

$$R(w(\mathbf{x}), a, w(\mathbf{y})) = r(w(\mathbf{y})). \tag{15}$$

The wrapped state MDP consists of the set of states $w(X) = \{w(\mathbf{x}) | \mathbf{x} \in X\}$, the same set of actions A as in the utilization state model, and reward function R and transition function P restricted to the wrapped states as described above. Since for any state $\mathbf{x} = w(\mathbf{x}) + \rho k\mathbf{u}$, for some $\rho \in \mathbb{Z}_+$, Theorem 1 applies, $V^*(w(\mathbf{x})) = V^*(\mathbf{x})$. It follows that $\pi^*(w(\mathbf{x}))$ on wrapped state $w(\mathbf{x})$ is the optimal action at each equivalent state in the utilization state model.

Bounding the State Model: The wrapped states are a subset of the utilization states. The utilization states are all of the integer-valued points in the positive octant of n-dimensional space. The wrapped states consist of all of the utilization states $\mathbf{x}$ with at least one component $x_a < ku_a$. This is equivalent to omitting any utilization state $\mathbf{x}$ in which for every action, $x_a \geq ku_a$. However, there are still infinitely many wrapped states. In order to work in the wrapped space, we need to introduce some mechanism to bound the number of states in our model.

As discussed above, the wrapped state model consists of possible displacements from the utilization ray among integer-valued states. In states that are sufficiently far from the target utilization we expect that the greedy policy, of choosing the action that is expected to put the state into a system nearest target utilization, is a suitable proxy for the optimal policy. Our strategy for bounding the state space is to truncate it to include only states that are not too far from target utilization for scheduling to be straightforward.

In order to formalize this distance threshold, we make an additional assumption about thread behavior. We assume that each thread cannot exceed some maximum run time on a single execution. In particular, we assume that each thread's run-time distribution has bounded support in $[0, \beta_a]$, so that $\sum_{t=0}^{\beta_a} P_a(t) = 1$. This implies that $P_a(t) = 0$ for any $t > \beta_a$, preventing the system from transitioning to an arbitrarily poor state after a single thread execution.

Given these run-time distribution bounds, we choose the size of our state space to accommodate some number of decisions without reaching the state space boundaries introduced by truncation. For example, any sequence of λ actions can not reach wrapped states outside of the region $\prod_{a=1}^{n}[0, \lambda\beta_a]$. We construct the bounded state space to accommodate λ steps from any wrapped state in $\prod_{a=1}^{n}[0, ku_a]$ by retaining any wrapped state in the region $\prod_{a=1}^{n}[0, ku_a + \lambda\beta_a]$. An action that would take the system out of this region in the wrapped model instead causes a transition to an absorbing states on the boundary. This is illustrated in Figure 3 on the example system shown in Figures 1 and 2. The transition function in this bounded state model is the same as the wrapped model transition function for states that cannot reach an absorbing state. If $w(\mathbf{x})$ is a wrapped state such that $w(\mathbf{x} + t\Delta_a)$ is an absorbing state (i.e., if $x_b + t\delta_{ab} = ku_b + \lambda\beta_b$), then the probability of moving into that absorbing state is $P(w(\mathbf{x} + t\Delta_a)|w(\mathbf{x}), a) = \sum_{t'=t}^{\beta_a} P_a(t')$. Determining whether or not λ is sufficiently large to retain the optimal policy for the original utilization state model is still an open question, although in practice $\lambda \geq 2$ appears to be sufficient.

We have implemented an approach that enumerates the states in this bounded state model by performing a breadth-first expansion of the state space rooted at $(0, 0, \ldots, 0)$. It is probable that more space-efficient methods for representing this state space are possible. A value iteration update to a single state consists of evaluating

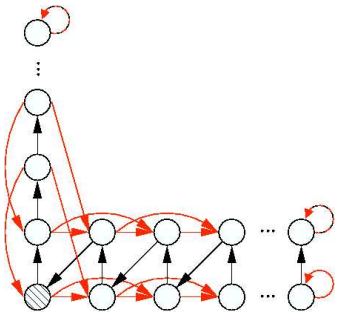

Figure 3: The wrapped state model from Figure 2 truncated with absorbing states. Vertices represent equivalence classes and edges represent transitions for thread one (red) and thread two (black).

$$Q(w(\mathbf{x}), a) = \sum_{t=0}^{\beta_a} P(\mathbf{y}_{t,a}|w(\mathbf{x}), a)\left[r(\mathbf{y}_{t,a}) + \gamma V(\mathbf{y}_{t,a})\right]$$

then maximizing over actions to find the new value $V(w(\mathbf{x}))$, where $\mathbf{y}_{t,a} = w(\mathbf{x} + t\Delta_a)$ (unless $w(\mathbf{x} + t'\Delta_a)$ is an absorbing state for $t' < t$, in which case $\mathbf{y}_{t,a} = w(\mathbf{x} + t'\Delta_a)$). Computing $P(\mathbf{y}_{t,a}|w(\mathbf{x}), a)[r(\mathbf{y}_{t,a}) + V(\mathbf{y}_{t,a})]$ for a given t and a consists of computing the successor state ($\Theta(n)$), its reward ($\Theta(n)$), and its value ($O(n)$ to compute the hash key, and expected $O(1)$ time to perform the lookup), for a total of $O(n)$ time per t and a. If all β_a are bounded above by β, this gives us an overall time complexity per state of $O(n^2\beta)$. This is repeated for each state in the bounded, wrapped state model until the change between iterations is sufficiently small.

There does not appear to be a simple expression for the number of states in this bounded state model. The number of utilization states that satisfy the step-based bounds is $\prod_{a=1}^{n}(\lambda\beta_a + ku_a)$. This is the number of integer states in the closed box $\prod_{a=1}^{n}[0, ku_a + \lambda\beta_a]$. Periodicity allows us to ignore any of the states in this box that are also in the closed box $\prod_{a=1}^{n}[ku_a, \lambda\beta_a + ku_a]$, so the number of states is

$$\prod_{a=1}^{n}(\lambda\beta_a + ku_a) - \prod_{a=1}^{n}\lambda\beta_a.$$

The most significant terms in this formula are $k^n \prod_{a=1}^{n} u_a$ and $ku_b\lambda^{n-1}\prod_{a\neq b}\beta_a$. We have eliminated the dependence on a termination time from the bounded utilization state model described above. There is still an exponential dependence on the number of threads, which we intend to address in future work.

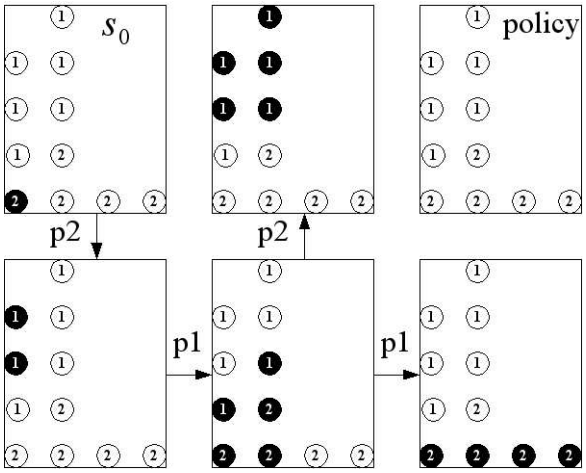

Figure 4: First states produced from the initial state, s_0, using aggregates of discrete utilization state as verification states. P_1 is supported on [3,5] and P_2 on [2,3]. The current verification state is shown by the darkened circles.

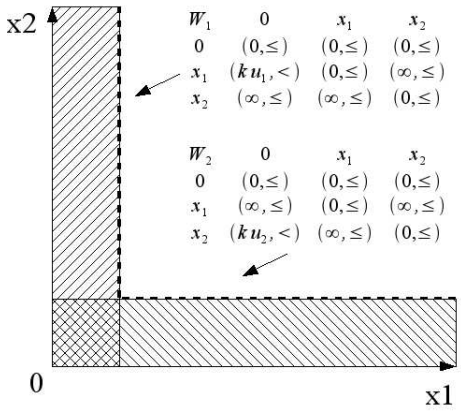

Figure 5: General form of the DBMs needed to encode the unbounded wrapped space. W_i represents the region satisfying $x_i < ku_i \wedge \bigwedge_{j=1}^{n} 0 \leq x_j$ where k is the minimum period given the system's target utilization $\mathbf{u}$.

5 Verification

Let π be a policy computed by the methods described in Section 4. Given this policy an obvious question is: what exactly can a system using this policy do? A more rigorous formulation of this question is: what are the possible set of thread executions of a system using this scheduling policy? To answer these and similar questions formally, we have developed a verification approach for the wrapped state space described in Section 4, that is suitable for model checking. Model checking works by keeping a set of visited states, initially equal to some subset of possible system states. A state s is selected from the set of visited states. The set of successor states given an action, $succ(s, a)$ is then calculated and added to the set of visited states. This process is repeated until a fixed point is reached where no new states are visited. In this section we present techniques for exhaustively enumerating the state space reachable using scheduling policy π. This state space can then be queried for properties such as safety, liveness, or composibility with other systems.

In our previous work [1], we had to contend with an unbounded number of states, which meant that no fixed point existed. Using the state space wrapping technique described in Section 4, the reachable set of utilization states under π is bounded if the run times of the individual threads are bounded. This suggests a simple method of model checking this system: use *sets of utilization states* as verification states. We therefore define $succ(s, a) = \{\{w(\mathbf{x} + t\delta_a) | w(\mathbf{x}) \in s,\ \pi(w(\mathbf{x})) = a,\ t \in [\alpha_a, \beta_a]\}\}$. Notice that in this case the set $succ(s_i, a)$ has only one member. It is the set that contains all utilization states reachable from each utilization state in s by taking action a. The thread runs for some time bounded by $[\alpha_a, \beta_a]$ (the region over which P_a, the run time distribution, has support). An example state space exploration is shown in Figure 4.

Our previous approach to enumerating the possible verification state space resulting from a given policy was to use a finite history [1]. That approached suffered from the fact that as the finite history grew in size, the resulting verification state space grew exponentially. Our space wrapping approach allows us to circumvent this problem, and greatly reduces the size of the state space enumerated, and consequently the time it takes to calculate it. In our previous approach, using a history size of 20 resulted in a state space with 2883441 states and 3604283 transitions which took 10 hours 47 minutes to enumerate. With our new approach the exploration takes less than a second and the state space has 201 states and 398 transitions. Both these experiments were done using the IF model checker.

Verification Using Continuous States: While an improvement on our previous techniques, further refinements to this approach are needed. Two concerns are that the space needed to represent a state is (1) equal to the size of the wrapped space, which is exponential in the number of threads, and (2) sensitive to the time quanta used in constructing the wrapped state. To represent verification states efficiently we represent them as continuous regions in the utilization space using difference bound matrices (DBMs) [21] which capture constraints between the values of variables. In a DBM each variable is bounded by formulas of the form $x_i - x_j \prec c$ where x_i and x_j are variables, c is an integer, and $\prec$ is either $<$ or $\leq$. To encode this particular formula in a DBM D, we would set the entry $D_{i,j} = (c, \prec)$. To measure distance from the origin a special variable is introduced whose value is always set to zero.

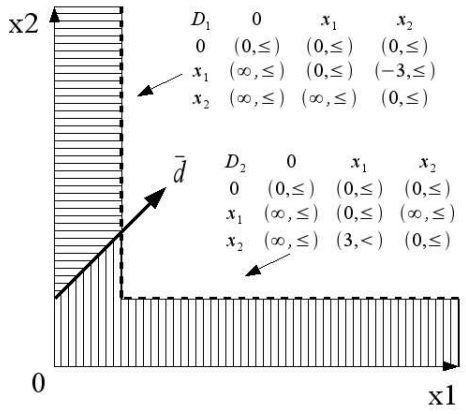

Figure 6: An example continuous two thread policy and the DBMs representing each decision region. D_i represents the region where $\pi(w(x)) = i$.

Each verification state is labeled with a DBM (specifically a *canonical* DBM) that bounds the values of $x_1, x_2, ...x_n$ in the utilization space.

In [22] two common operations on DBMs are described in detail: tightening and intersection. Tightening a DBM converts it to a canonical form, and allows us to check easily if the set of utilization states that satisfy the constraints is non-empty. Intersection returns a DBM that describes the region that is the intersection of the two regions, which the intersected DBMs describe.

We define two new operations: run and $wrap$. The run operation captures the effect of running thread a. The $wrap$ operation is analagous to the wrapping function described for individual utilization states but takes an additional parameter $\rho \in \mathbb{Z}_+$.

To define $D' = run(D, a, \alpha_a, \beta_a)$ let $D_{i,j} = (c, \prec)$. If $i \neq a, j \neq a$ then $D'_{i,j} = (c, \prec)$. If $i = a, j = a$ then $D'_{i,j} = (c, \prec)$. If $i \neq a, j = a$ then $D'_{i,j} = (c - \alpha_a, \prec)$. If $i = a, j \neq a$ then $D'_{i,j} = (c + \beta_a, \prec)$.

To define $D' = wrap(D, \rho)$, let $D_{i,j} = (c, \prec)$ and k be the minimum period for the system target utilization $\mathbf{u}$. If $i = 0, j = 0$ then $D'_{i,j} = (c, \prec)$. If $i \neq 0, j = 0$ then $D'_{i,j} = (c - \rho k u_i, \prec)$. If $i = 0, j \neq 0$ then $D'_{i,j} = (min(0, c + \rho k u_j), \prec)$. If $i \neq 0, j \neq 0$ then $D'_{i,j} = (c - \rho k u_i + \rho k u_j, \prec)$.

In addition, we need DBMs representing salient features of both our policy and our wrapped space. Figure 5 shows the general form of DBMs needed to represent the wrapped space while Figure 6 shows decision regions encoded as DBMs. While the description of the wrapped space as a disjunction of DBMs is straightforward, it is not reasonable to expect that the decision regions of an arbitrary policy created by the procedure in Section 4 can be directly expressed as a DBM. For instance, in the two thread case, the boundary separating the decision regions is parallel to the ray encoding the system's target utilization $\mathbf{u}$. If $\mathbf{u}$ does not have slope 1, neither will the boundary betweeen decision regions, and thus they cannot be expressed as DBMs. Two options exist: (1) to build a convex shell around the actual decision region that can be expressed as a DBM (or a disjunction of DBMs), or (2) to force the target utilization $\mathbf{u}$ to have slope 1 by skewing the utilization space and thread run time distributions. The first is more generally applicable, but may be time consuming to calculate and by necessity is a coarser approximation. When possible the second method is preferred , provided the skewing does not alter the system semantics.

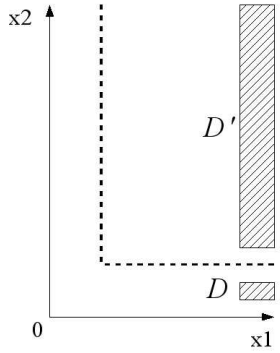

Figure 7: Running thread 2 transforms DBM D into D'.

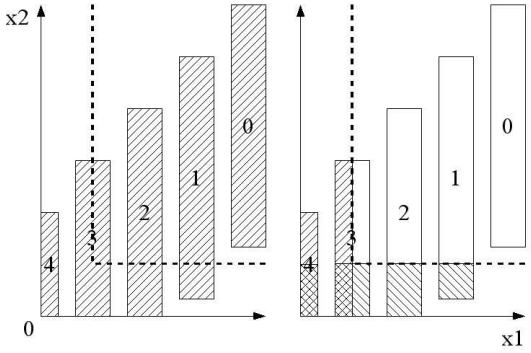

Figure 8: Results of applying the $wrap$ operator to a DBM for different values of ρ, and the intersection of the resulting DBMs with W_i for each dimension in the utilization space.

The set of verification states, $succ(s, a)$, is obtained by the following steps, each defined in terms of the set of DBMs, s_{in}, intially containing the DBM labeling state s, but afterward equal to the set s_{out} defined in the previous step. After each step each DBM in s_{out} is tightened, and if the set of possible utilization states that satisifies the constraints is empty, the DBM is removed from s_{out}:

1. $s_{out} = \{s \cap D_a | s \in s_{in}\}$.
2. $s_{out} = \{run(s, a, \alpha_a, \beta_a) | s \in s_{in}\}$.

3. $s_{out} = \{wrap(s, \rho) | \rho \geq 0,\ s \in s_{in}\}$.
4. $s_{out} = \{s \cap W_i | 1 \leq i \leq n,\ s \in s_{in}\}$.

Step 2 is shown in Figure 7, steps 3 and 4 in Figure 8. The above procedure produces a set of DBMs which label the verification states in $succ(s, a)$.

6 Conclusions and Future Work

In this paper we have presented new techniques for efficiently creating scheduling polices based on a given utilization specification, and then building a state space over which system properties can be verified. These techniques constitute advances in the representation and use of both system and verification states. We show that system states can be compactly represented via a novel wrapping mechanism enabled by self-similarity of costs and state transitions. This allows us to concisely capture sets of these states for model checking via difference bound matrices in order to express both discrete and continuous time semantics.

This approach still suffers from the curse of dimensionality in that the number of system states grows exponentially with the number of threads. A hierarchical representation of state can be obtained by separating threads into groups that can be scheduled separately. This approach, which we plan to pursue as future work, may substantially reduce the number of system states.

Another area of planned future work is systems with continuous state spaces. We are able to verify policies over such spaces, but actually computing optimal scheduling policies in these spaces remains an open challenge. In future work we intend to address this shortcoming by looking at the limit behavior of the discrete methods described in this work as the time step size decays.

In addition we plan to expand our techniques to handle richer system models including: unknown distributions of thread run times, scheduling with preemption, threads that may block for I/O, and other forms of resource sharing among threads.

References

[1] T. Tidwell, R. Glaubius, C. Gill, and W. D. Smart, "Scheduling for Reliable Execution in Autonomic Systems," in 5^{th} *International Conference on Autonomic and Trusted Computing (ATC '08)*, (Oslo, Norway), June 2008.

[2] ARINC Incorporated, Annapolis, Maryland, USA, *Document No. 653: Avionics Application Software Standard Inteface (Draft 15)*, Jan. 1997.

[3] W. Mark Vanfleet and Jahn A. Luke and R. William Beckwith and Carol Taylor and Ben Calloni and Gordon Uchenick, "MILS: Architecture for High-Assurance Embedded Computing." `http://www.stsc.hill.af.mil/crosstalk/2005/08/0508Vanfleet_etal.html`, Crosstalk: the Journal of Defense Software Engineering, August, 2005.

[4] W. Martin, P. White, F. S. Taylor, and A. Goldberg, "Formal construction of the mathematically analyzed separation kernel," in *ASE '00: Proceedings of the 15th IEEE international conference on Automated software engineering*, (Washington, DC, USA), p. 133, IEEE Computer Society, 2000.

[5] J. W. S. Liu, *Real-time Systems*. New Jersey: Prentice Hall, 2000.

[6] Y. Zhang, C. Lu, C. Gill, P. Lardieri, and G. Thaker, "Middleware support for aperiodic tasks in distributed real-time systems," in *RTAS '07: Proceedings of the 13th IEEE Real Time on Embedded Technology and Applications Symposium*, (Washington, DC, USA), pp. 497–506, IEEE Computer Society, Apr. 2007.

[7] Goyal, Guo, and Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," in 2^{nd} *Symposium on Operating Systems Design and Implementation*, USENIX, Oct. 1996.

[8] Regehr and Stankovic, "HLS: A Framework for Composing Soft Real-time Schedulers," in 22^{nd} *IEEE Real-time Systems Symposium*, (London, UK), Dec. 2001.

[9] Regehr, Reid, Webb, Parker, and Lepreau, "Evolving Real-time Systems Using Hierarchical Scheduling and Concurrency Analysis," in 24^{th} *IEEE Real-time Systems Symposium*, (Cancun, Mexico), Dec. 2003.

[10] T. Aswathanarayana, V. Subramonian, D. Niehaus, and C. Gill, "Design and performance of configurable endsystem scheduling mechanisms," in *Proceedings of 11th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS)*, 2005.

[11] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine, "Decidable integration graphs," *Information and Computation*, vol. 150, no. 2, pp. 209–243, 1999.

[12] T. Tidwell, C. Gill, and V. Subramonian, "Scheduling induced bounds and the verification of preemptive real-time systems," Tech. Rep. WUCSE-2007-34, Computer Science and Engineering Department, Washington University in St.Louis, 2007.

[13] H.-M. Huang and C. Gill, "Modeling timed component-based real-time systems," Tech. Rep. WUCSE-2008-1, Computer Science and Engineering Department, Washington University in St.Louis, 2008.

[14] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, pp. 196–210, March 1962.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[16] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "On the use of hybrid reinforcement learning for autonomic resource allocation," *Cluster Computing*, vol. 10, September 2007.

[17] M. L. Littman, N. Ravi, E. Fenson, and R. Howard, "Reinforcement learning for autonomic network repair," in *Proceedings of the 1st International Conference on Autonomic Computing (ICAC 2004)*, pp. 284–285, 2004.

[18] S. Whiteson and P. Stone, "Adaptive job routing and scheduling," *Engineering Applications of Artifi cial Intelligence*, vol. 17, pp. 855–69, oct 2004.

[19] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Interscience, 1994.

[20] S. P. Singh and R. C. Yee, "An upper bound on the loss from approximate optimal-value functions," *Machine Learning*, vol. 16, no. 3, pp. 227–233, 1994.

[21] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Proceedings of the international workshop on Automatic verifi cation methods for fi nite state systems*, pp. 197–212, Springer-Verlag LNCS 407, 1990.

[22] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. MIT Press, 1999.