RESEARCH ARTICLE OPEN ACCESS

Improving and extending Utility-Aware Scheduling Heuristics for Soft Real-time Systems with Stochastic Running Times

Mohammadsadegh Mohagheghi [1], Behrang Chaboki [2]
Department of Computer science
Vali-e-asr University of Rafsanajn, Rafsanjan - Iran

ABSTRACT
Time utility functions have been widely used to describe the deadline constraints of real-time embedded systems. Utility-aware scheduling is an extension to the classic scheduling problem of real-time systems that considers utility functions and its objective is to maximize the total utility that is accrued by completing the activities. Because of policy storage requirements, heuristic algorithms should be used for this class of scheduling problems. Utility Accrual Packet Scheduling Algorithm (UPA) is a well-known heuristic proposed for this problem. In this paper, we improve and extend UPA heuristic for scheduling utility-aware soft real-time systems with stochastic execution times. Our improvement uses the information of future arrival jobs in a periodic real-time system to avoid some useless selections of the UPA heuristic. We also extend UPA for energy harvesting utility-aware real-time systems.
Keywords: — Scheduling, Soft real-time Systems, Utility Functions

I. INTRODUCTION

There are many application domains for dynamic, real-time embedded systems. Soft time constraints are included in many of these systems. Time Utility Functions (TUFs) are a way for precisely specifying the semantics of soft time constraints [1]. In this sense, the system can gain a utility according to the time of completing a job [2], [3]. Penalties are used as utilities with negative values. These functions show the utility of completing jobs as a function of time and are considered as a generalization for the concept of classical deadlines [4]. The optimization - maximizing (or minimizing in) the expected sum of the jobs attained utilities (or penalties) - for the accrued utility from the jobs of a system is a base for the scheduling criteria of TUF-based soft real-time systems. In the literature, such criteria are called Utility Accrual (or UA) criteria [5] and UA scheduling algorithms are those that consider UA criteria and should compute the optimal expected values and policies. One challenge of this class of scheduling problems is that the needed space for storing the information of the optimal policy grows exponentially in the number of tasks of the model. Because embedded systems usually have a limitation for storing any type of information, the exponential space complexity of storing the optimal policy can limit the possibility to use an offline optimal method for this class of problems. In addition, this class of scheduling problems are NP-complete [5]. Because of the memory limitation and the complexity of the problem, some heuristics should be used to compute near optimal scheduling [3], [6].

There are considerable applications and real case studies for UA scheduling problems including Ethernet packet scheduling [14], robotics [32] and database services [33]. Energy efficient UA scheduling methods are used in cloud computing [34], [35] and wireless sensor networks [36], [37].

Many real-time embedded systems (such as mobile robotics) have some features that complicate their scheduling problem and as a result their scheduling algorithms should consider non-preemptable jobs [7]. The ability of these systems to interact with the physical world is an important reason for this complication. This interaction sometimes leads to nondeterministic job durations and uncertain behaviors. A standard approach for including uncertain job durations is worst-case execution time (WCET) analysis that provides statistical assurance on timeliness behavior [6]. For better understanding of the behavior of the system and more accurate scheduling, UA scheduling algorithms on the other hand can consider probabilistic distribution of job durations [4], [8].

An MDP-based approach has been proposed to compute the optimal policy and expected UA [7] for embedded systems with probabilistic distribution of job durations. However, this approach has also the same limitation of the standard UA scheduling problems. It usually needs high amount of memory to store the optimal policy and cannot be used as an offline approach. An alternative solution is to use appropriate heuristics with lower memory and time complexity at run-time to approximate the value optimal policy of UA scheduling problems [4]. Utility Accrual Packet Scheduling Algorithm (UPA) and pseudo UPA are two heuristics that has been proposed for approximating optimal values of this problem and are applicable for a variety of time utility functions [9]. In this paper we use an approach to improve the performance of previous heuristics. The idea of our approach is to consider future job arrivals to the system and their utilities and their impact on the expected utility of the system. One drawback of the UPA and pseudo α heuristics is that they always select a job from the ready queue while the selected job may return low utility to the system and also postpone the execution of the next jobs with relatively higher utilities. In this case, the optimal choice may be to let the system to be idle for a short time in order to utilize from the future job.

Because batteries are primary energy sources of many embedded systems, a lot of studies have been done in recent years in order to increase the energy-efficiency of battery based embedded systems. In this paper, we also extend UA heuristics to consider battery based systems and propose some approaches for approximating optimal expected values under energy constraints of these systems. As a result, our work has two main contributions: Improving the performance of the UPA and pseudo α heuristics for UA scheduling of the systems with probabilistic durations and extending these heuristics for the scheduling problems of the battery based energy harvesting systems. We use some simulations to compare the performance of our improvements to the performance of previous heuristics. This paper is organized as follows: In Section 2 we provide a brief review of the related works. Section 3 provides the definition of the problem and its task model. Section 4 reviews previous offline and online approaches for computing optimal UA schedulers. In Section 5 we propose our improvement to the UPA heuristic for the scheduling problem. In Section 6 we describe our approach for modeling battery based UA scheduling problem and describe our extension to previous heuristics for this problem. Section 7 covers experimental results and Section 8 concludes the paper and provides directions for future works.

II. RELATED WORKS

A main part of real-time embedded control systems have soft time constraints (besides hard) in the sense that missing a deadline does not have catastrophic results [10]. Jensen's time/utility functions [1] have been proposed to precisely specify the semantics of soft time constraints. It is known that optimal utility accrual scheduling problem is NP-hard [11] and heuristic algorithms are needed for real applications. Dependent Activity Scheduling Algorithm [12], Gravitational Task Model [13] and UPA [14] are some of these heuristics that are restricted to special shapes of time-utility curves. The Generic Utility Scheduling algorithm (GUS) can be used for arbitrarily shaped time/utility functions [5, 15]. While it can propose near optimal policies, its time complexity is in $O(n^3)$ and is not applicable for most of real case studies. History-Cognisant Time-Utility-Functions (HCUF) is an extension to the Earliest Deadline First (EDF) algorithm that is proposed for UA scheduling of overloaded real-time systems [3]. Profit and Penalty Aware (PP-aware) Scheduling [16], [17] and Online Real-Time Service-Oriented Task scheduling [18] consider those systems with both positive and negative utility functions and variable task scheduling times. These two methods propose online scheduling algorithms for maximizing system's total accrual benefit. Most of these works, consider WCET for analyzing tasks with variable running times. However another approach is to consider stochastic uncertainty for UA scheduling. Several methods have been proposed to provide stochastic assurances on timeliness behavior [19], [20]. Tidwel has proposed an MDP-based approach for computing the optimal stochastic UA scheduling [7]. Although this approach can determine the optimal policy of the related problem, its time and space complexity is exponential in the number of tasks of the problem. Because of its time complexity, it is not possible to use this method at run time. In addition the amount of memory that is needed to store the information of the optimal policy is usually more than the actual memory that is available in many real-time embedded systems and we have to use an efficient online approach. It is the main motivation of our work: we cannot perform it as a pre-computation to determine the optimal (or near optimal) policy. Instead, as an alternative approach we can use a good heuristic with low memory and time overhead to estimate a near-optimal policy at run time. UPA α and Pseudo α are two heuristics that have proposed for approximating optimal policies for this class of scheduling problems [4]. A decision tree-based approach was proposed to store the value-optimal policy (or an approximation of it) as another alternative for dealing with the memory limitation problem. In this way, the MDP-based method determines the optimal policy and a decision tree can be used to store the actions of the optimal policy [4]. This approach can be considered as a static scheduling method and its accuracy depends on the size of the optimal policy and the size of its corresponding tree.

Model checking has been widely used as a complementary approach to schedule synthesis [21]. Scheduling with Timed Automata (TA) has been studied for both hard and soft real-time systems [22]-[26]. Duration probabilistic automata, an extension of TA was proposed for computing the expected performance of a given scheduling policy [27]. In the case of stochastic UA scheduling, TA can be used for generating cost-optimal schedules [28].

Energy efficient real-time scheduling has been studied in recent years for different types of embedded systems. In the case of UA scheduling, Energy-efficient methods and heuristics have been proposed in [29] which can also be used for a stochastic version of the problem. A probabilistic scheduling approach for energy efficient task assignment was proposed in [30] that are useful for both hard and soft real-time systems. TA has been also used for verifying feasibility and schedulability test of the real-time scheduling problem under energy constraints [31].

III. PROBLEM DEFINITION AND TASK MODEL

In this section we define the UA scheduling problem according to the notations of [7]. Most definitions are standard and are proposed as in [1], [4], [7]. A periodic real-time system is defined as a set of n tasks $(T_i)_{i=1}^{n}$ with mutual exclusive use of a single common resource. Each task T_i contains an infinite sequence of non-preemptable jobs. We use $J_{i,j}$ to denote the jth job in T_i and p_i for the period of this task. A new job of T_i is added to the ready queue every p_i time units and $r_{i,j}$ is used as the release time of $J_{i,j}$. Note that for every two subsequent jobs $J_{i,j}$ and $J_{i,j+1}$, we have $r_{i,j+1} = r_{i,j} + p_i$ and for every T_i we have $r_{i,0} = 0$. Each job has a deadline $d_{i,j}$ that is the time which the job should be completed before it. For each T_i a relative deadline τ_i is

defined relative to the release time: $\tau_i = d_{i,j} - r_{i,j}$. In this paper for all tasks we have $\tau_i \leq p_i$.

Jobs remain in the ready queue until they are scheduled to start to run or they miss their deadline, i.e. they cannot finish their work before their deadline. After finishing each job, the scheduler of the system chooses either to run a job from the ready queue or to idle the system for a unit of time. For each job of a task T_i a stochastic execution time is defined according to a probability distribution function D_i where $D_i(t)$ is the probability of the job's execution time being exactly t. The minimum and maximum execution time of the jobs of each task T_i are shown by l_i and w_i. Consequently for each $t < l_i$ and $t > w_i$ we have $D_i(t) = 0$. For each task T_i a time utility function $U_i(t)$ is associated that maps the time elapsed since $r_{i,j}$, denoted by t, to a real number that determines the utility of completing $J_{i,j}$ at time $r_{i,j} + t$. In this paper we consider linear drop utility functions (Fig 1, as is proposed in [7]) and we define $U_i(t) = 0$ for every time $t > \tau_i$.

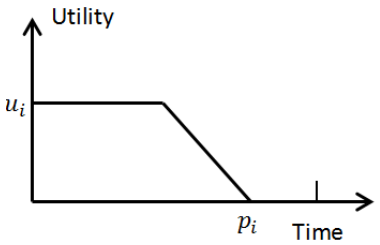


Fig 1- A Linear Drop Utility Function

In hard real-time systems all jobs have to meet their deadline and the problem is to find a scheduling for which no job misses its deadline. In soft real-time systems, on the other hand, a missed deadline of a job means that the system keeps working at a reduced degree of performance. In this case the scheduler should select jobs in order to maximize the expected value of utilities in long run. The processor workload usually affects the performance of the (soft or hard) real-time system. It is defined as the amount of computation (processor demand) that is required in a unit of time [10]:

Definition 1: The processor demand $g_i(t_1, t_2)$ of each task T_i of the system is defined as the expected computation time requested by those jobs of T_i that their release times and deadlines are within $[t_1, t_2]$. Considering probability distribution of execution times, we have $g_i(t_1, t_2) = \sum_{r_{i,j} \geq t_1, d_{i,j} \leq t_2} ExpTime(T_i)$ where $ExpTime(T_i) = \sum_{t=l_i}^{w_i} D_i(t).t$ is the expected execution time of jobs of T_i. Because the D_i function depends only on the time of the system and is the same for every job of the task, we use $ExpTime(T_i)$ in the definition of g_i. If D_i is different for some different jobs, we should use $ExpTime(J_{i,j})$. The total processor demand of a system with n tasks in an interval $[t_1, t_2]$ is defined as $g(t_1, t_2) = \sum_{i=1}^{n} g_i(t_1, t_2)$ and the processor workload is defined as $\rho(t_1, t_2) = \frac{g(t_1, t_2)}{t_2 - t_1}$. A system is said to be in overload condition if there exists an interval of time $[t_a, t_b]$ such that $\rho(t_a, t_b) > 1$. An overload condition means that in every hyper-period at least one job may miss its deadline. ■

Consider, for example, a system with two tasks where $p_1 = 6, \tau_1 = 5$ and $p_2 = 8, \tau_2 = 6$. For D_i we have $D_1(2) = D_1(3) = \frac{1}{2}$ and $D_2(2) = D_2(3) = \frac{2}{5}$ and $D_2(4) = \frac{1}{5}$. As a result $ExpTime(T_1) = 2.5$ and $ExpTime(T_2) = 2.8$. The hyper-period of these two tasks is $H = \gcd(6,8) = 24$. For the interval $[0,24]$ we have $g_1(0,24) = 4 \times 2.5 = 10$ and $g_2(0,24) = 3 \times 2.8 = 8.4$. The total processor demand in this interval is $g(0,24) = 18.4$ and the processor workload is $\rho(0,24) = \frac{18.4}{24} \simeq .77$

IV. OPTIMAL SOLUTION AND HEURISTICS FOR UTILITY AWARE SCHEDULING

In this section we review the definition of Markov Decision Processes and previous works for computing optimal expected utility of soft real-time systems [7] and some proposed heuristics for this problem [9].

A. Markov Decision Process

A Markov Decision Process (MDP) [38] is a four-tuple (S, A, P, R) where

- S is a finite set of states
- A is a finite set of actions
- P is a probabilistic transition function. It determines the probability of going from a source state s to a destination state s' by an action a. We use $P(s, a, s')$ for showing this probability.
- $R$is a reward function. $R(s, a, s')$ determines the immediate reward for taking an action a from state s and going to the state s'.

MDPs are used for modelling decision making of systems under uncertainty. The system is considered to execute its actions in discrete steps. At each step$t$, it is in one state $s_t \in S$. It can select one action $a_t \in A$ and perform a transition to the next state s_{t+1} with the probability $P(s_t, a_t, s_{t+1})$. This transition incurs the reward of $R(s_t, a_t, s_{t+1})$to the system.The number of steps for which rewards of the MDP are accumulated is called its horizon. According to this concept we have two types of MDPs: finite horizon vs infinite horizon MDPs. For finite horizon MDPs a set $G \subset S$ of sink goal statesare defined. The system continues until reaching a goal state and should terminate in any of goal states [38,39].In this case the probability of reaching a goal state is needed to be one for long runs.

The entire document should be in Times New Roman or Times font. Type 3 fonts must not be used. Other font types may be used if needed for special purposes.

For every state $s \in S$ and action $a \in A$ we can define the (expected) reward according to the probabilistic transitions and their rewards (and independent of time step):

$$R(s,a) = \sum_{s' \in S} P(s,a,s').R(s,a,s') \quad (1)$$

A policy $\pi: S \to A$ is a mapping that determines which action should be selected for execution at each state. For the case of finite horizon and for every state $s \in S$ the value of a policy is the expected sum of rewards of the MDP until reaching one of goal states:

$$V^{\pi}(s) = E\{\sum_{k=0}^{g} R(s_k, \pi(s_k), s_{k+1}) | s_0 = s, s_g \in G, \forall k < g \; s_k \notin G\} \quad (2)$$

where $\pi(s_k)$ is the action that is selected by the policy π.For the case of infinite horizon, we cannot use this equation because $V^{\pi}(s)$ may diverge for any policy π.The alternative approach for infinite horizon MDPs is to use a discount factor γ [38]. In this case the value of a policy is defined as:

$$V^{\pi}(s) = E\{\sum_{k=0}^{\infty} \gamma^k . R(s_k, \pi(s_k), s_{k+1}) | s_0 = s \} \quad (3)$$

An equivalent definition for V^{π} in the case of finite horizon is the solution to the following linear system:

$$V^{\pi}(s) = \sum_{s' \in S} [P(s, \pi(s), s') \times V^{\pi}(s') + R(s, \pi(s), s') \quad (4)$$

An important problem for MDPs is to find the optimal policy, the policy that maximizes (or minimizes) the expected reward for the model. Let $V^*(s)$ denote the value of the optimal policy. Then $V^*(s)$ should satisfy the Bellman equation [40]:

$$V^*(s) = 0 \quad if \; s \in G$$
$$V^*(s) = \max_{a \in A} \{R(s,a) + \sum_{s' \in S} P(s,a,s').R(s,a,s')\}$$

otherwise (5)

In this case the optimal policy can be computed as:

$$\pi^*(s) = argmax_{a \in A} \{R(s,a) + \sum_{s' \in S} P(s,a,s').R(s,a,s')\}\} \quad (6)$$

Similar definitions are used for infinite horizon MDPs that also use a $\gamma < 1$ discount factor for converging to a fixed point solution. [7], [38]. Value iteration [40] and policy iteration [38] are two well-known methods for computing optimal values and policies of an MDP. In this paper we use the Gauss-Seidel value iteration [38], [39] for computing optimal values and policies of MDPs.

B. Using MDPs for Modeling Stochastic Utility-Aware Task Scheduling

We review the method of [7] for using MDPs to compute the optimal stochastic UA scheduling policy. This method considers a discrete semantic of time. We first describe how to define each component of an MDP according to a stochastic UA scheduling problem. Next we explain which type of MDPs (finite vs. infinite horizon) should be used. For modelling this problem, each state of the MDP should capture two types of information: The system time (t_{system}) and a Boolean vector that shows the existence of every job in the ready queue, i.e. we have $s = (t_{system}, b_1, \dots, b_n)$. The value of b_i is true if T_i has a job in the ready queue at time t_{system} and is false otherwise.In general the value of t_{system} for real-time embedded systemsis not bounded and the system can continue forever. This causes the MDP of the system to have infinite number of states but the idea of wrapped UA scheduling MDP can limit the number of states to a finite number. One can consider the hyper-period$H$ of the tasks of the system, that is defined as the least common multiple of the task periods, to limit the analysis of the behaviour of the system to one hyper-period. For every two states $s = (t, b_1, \dots, b_n)$ and $s' = (t', b_1, \dots, b_n)$, if the difference between t and t' is a multiple of $H (t' - t = k \times H, k \geq 1)$ these two states have the same outgoing probabilistic transitions and the same related rewards. As a result, the optimal policy should select the same actions for both states and it is enough to consider only those states for which $t_{system} < H$.

At the start point of the system or after finishing each job, the scheduler should select one job from the ones in the ready queue or let the system to be idle for one step of time. According to this selections the set of actions of the MDP are defined:a_i for selecting J_i from the ready queue and a_{idle} to advance time in the system by one unit. Note that the scheduler can select a_i for any state $s = (t_{system}, \dots, b_i, \dots)$ only if the value of b_i is true for this state. For each state, the set of actions that can be selected by the scheduler are called enabled actions of that state. Every state s has at least one enabled action (a_{idle}) and at most n+1 enabled ones $(a_{idle}, a_1, \dots, a_n)$.

The transition function $P(s,a,s')$ is the probability of going from s to s' with the action$a$and is defined according to the t_{system} component of s, s' and the probabilistic distribution of execution time of the job that is selected by the action a .Let $s = (t_{system}, b_1, \dots, b_n)$ and $s' = (t'_{system}, b'_1, \dots, b'_n)$ and $a = a_i \neq a_{idle}$, then $P(s,a,s') = D_i(t)$ if $b_i = true$ and$(t_{system} + t) \; mod \; p_i < \tau_i$ and $t'_{system} = (t_{system} + t) mod \; H$. This transition means that in the state s with the time component t_{system} a job of the task T_i is in the ready queue and the scheduler can select this job. The execution time is t units of time with probability of $D_i(t)$ and as a result the time component of the destination state s' is $t_{system} + t$. An important condition for this transition is that the job should be finished before its deadline that means $(t_{system} + t) mod \; p_i$ should be less than τ_i. The time component of the source and destination states and the period of each task determine the values of b' variables. For a_{idle} we have $P(s, a_{idle}, s') = 1$ where $t'_{system} = (t_{system} + 1) \; mod \; H$.

Definition of the reward function of the MDP is similar to the definition of the transition function. If $P(s,a,s') = 0$ or $a = a_{idle}$ then $R(s, a_{idle}, s') = 0$, otherwise $R(s, a_{idle}, s')$ is defined as the utility density of J_i that is $U_i(\tau - r_i)/(\tau - r_i)$ [4] where τ is t_{system} and r_i the release time of the current job of

T_i can be computed using t_{system} and p_i. For any state s and action a_i the immediate reward can be defined as a function of the current time t_{system}, the release time r_i of J_i and the utility function U_i and also D_i as:

$$R(s,a_i) = \sum_{d=1}^{w_i} \frac{D_i(d).U_i(d+(t_{system} \bmod p_i))}{d} \quad (7)$$

Consider $s = (t_{system} = 20, b_1 = true, b_2 = true)$ as a state of the MDP for the previous example. The probabilities of outgoing transitions of this state with the action a_2 are $P(s,a_2,s') = P(s,a_2,s'') = \frac{2}{5}$ and $P(s,a_2,s''')\frac{1}{5}$ where $s' = (t_{system} = 22, b_1 = true, b_2 = false)$, $s'' = (t_{system} = 23, b_1 = false, b_2 = false)$ and $s''' = (t_{system} = 24 \bmod H = 0, b_1 = true, b_2 = true)$. The value of b_2 in s' and s'' is $false$ that means the current job of T_2 has been selected and removed from the ready queue but this value is one in s''' because the next job of T_2 should be arrived in this state.The value of b_1 in s' is true because it is true in s and $t_{system} \bmod p_1 < \tau_1$ holds in this state but it does not hold in s'' that means $b_1$should be considered false in this state.

C. Space Complexity of MDP based Method for Stochastic Utility-Aware Task Scheduling

Every state of the MDP of a stochastic UA scheduling problem has a t_{system} component that ranges over $[0, H-1]$. In addition the model needs n Boolean variables $b_1, \dots, b_n$ where in general the value of each one is independent of the value of others. For each task T_i the value of b_i should be $true$ when $t_{system} \bmod p_i = 0$ but for other values of t_{system} this value can be $true$ or $false$. It means that an upper bound for the number of states of the MDP is $2^n \times H$. For each state s and action a_i the number of outgoing transitions is equal to the number of time units t for which $D_i(t) > 0$ that is bounded by $w_i - l_i + 1$. We have only one outgoing transition for a_{idle}.In the worst case the period of each task is equal to H and for each T_i we have $w_i - l_i + 1 = H$. This means that an upper bound for the number of transitions of the MDP is $2^n.(n+1).H$

D. Online Heuristics for Computing Optimal UA Schedulers

One drawback of MDP based approach for computing the optimal policy of a stochastic UA-based real-time system is that the system needs to store the optimal actions of each state for using at run-time. This approach can be much expensive for embedded systems that usually have limitations in their storage. Two alternative approaches have been proposed in [4]:

1- The scheduler can use an efficient heuristic (with low time complexity) at run-time
2- The scheduler can use MDP based approach as the pre-computation step anduse a decision tree to store an approximation of the optimal policy.

The first approach (that is also called an on-line scheduling) needs to use a heuristic with low overhead and near optimal policies. The second approach computes the optimal policy (as an offline scheduling) and use a binary tree to store an approximation of the optimal policy in the memory of the embedded system. To the best of our knowledge UPA shows the best performance for UA scheduling of systems with stochastic durations [9]. In this paper we propose a method for improving the performance of UPA and compare our approach with standard UPA [9], [14] , $pseudo\ \alpha$ [9], EDF [41] and greedy [7].

E. UPA heuristic for real-time systems with stochastic execution times

The Utility Accrual Packet Scheduling Algorithm [14] considers the utility density of every job in the ready queue and the impact of selecting a job on the performance of other jobs in the queue. It first selects the set of jobs that can finish before their deadline, and then sorts the remaining jobs by the pseudo slope of their utility function and finally it tries to maximize local utility by applying a method similar to the Bubble sort. Figure2 describes the UPA algorithm for systems with stochastic distributions. It works like the standard UPA but considers the stochastic distributions of the running times of jobs.$\Delta_{i,j}(t)$ determines the difference between the expected value of executing J_j immediately after J_i and the expected value of executing J_i immediately after J_j. As the result the algorithm switches the order of J_i and J_j if $\Delta_{i,j}(t)$ is negative. We use r_i for the release time of the current job of T_i.

Input: Q the ready queue of jobs and t_{system}.

Output: A near-optimal job

1. for each J_i in Q do
2. if $w_i + t_{system} > \tau_i$
   Drop J_i from Q
3. for each remaining job J_i
PseudoSlope(J_i) = $U_i(t_{system} - r_i)/(\tau_i - t_{system})$;
4- Sort the jobs in Q according to their PseudoSlope decreasingly
5- for k = 1 to |Q|
   5.1 InOrder = True;
   5.2 for i = 1 to |Q|
      5.2.1 j = i + 1;
      5.2.2 $\Delta_{i,j}(t) = \sum_{d_1=1}^{w_i} D_i(d_1).[U_i(t \bmod p_i + d_1) + \sum_{d_2=l_j}^{w_j} D_j(d_2).U_j(t \bmod p_j + d_1 + d_2)] - \sum_{d_1=l_j}^{w_j} D_j(d_1).[U_j(t \bmod p_j + d_1) + \sum_{d_2=l_i}^{w_i} D_i(d_2).U_i(t \bmod p_i + d_1 + d_2)]$
      5.2.3- if $\Delta_{i,j}(t) < 0$
         Swap the order of J_i and J_j in Q;
         InOrder = false;
      Endif
   5.3 if InOrder = True
      Break;
6- return J_1 as the selected job

Fig 2. UPA for stochastic UA Scheduling

The time complexity of an efficient implementation of this algorithm is $O(n^2.w)$ where w is the maximum WCET of all tasks [9]. This time complexity can be high for a dynamic heuristic for an embedded system and can cause a considerable overhead to the system. Two alternatives for this algorithm have been proposed [4]:

1- $UPA\ \alpha$ which considers only those jobs that can be finished before their deadline with probability of α.
2- $Pseudo\ \alpha$ that works like the $UPA\ \alpha$ but only considers the pseudo slope of timed utility functions

Although the worst case time complexity of $UPA\ \alpha$ is the same as UPA but in many cases it needs only to consider a small number of n tasks and as the result its overhead is less than the overhead of UPA. Time complexity of $Pseudo\ \alpha$ is $O(n^2)$ [9] that is more applicable for many real-time embedded systems.

V. IMPROVING UPA

Empirical studies show that the performance of greedy and UPA heuristics are near optimal for systems with medium and low load scenarios but their performance are not much good for the case of heavy load when the system is in overload condition. The main drawback of these heuristics for systems with high workload is that in every state of the system, they only consider the jobs that are in the ready queue but do not consider future jobs that are near to arrive to the system. In some cases selecting one job from the ready queue can cause some jobs that will release in future miss their deadlines. Consider the example in Section 3 with T_1 and T_2 and an additional task T_3 with $p_3 = 30$ and $\tau_3 = 20$ and $D_3(10) = D_3(11) = \frac{1}{2}$. Consider a state of the system where $t_{system} = 22$ and $J_{1,4}$ and $J_{2,3}$ have finished and the only job in the ready queue is J_3 that means $s = (t_{system} = 22, b_1 = false, b_2 = false, b_3 = true)$.UPA, EDF and greedy heuristics select J_3 for starting its computation but this decision can postpone the start of $J_{1,5}$and $J_{2,4}$. In fact these two jobs will miss their deadline if the scheduler selects$J_{3,1}$when $t_{system} = 22$. The optimal selection depends also on the value of TUFs of these three tasks. If for example U_i is the same for all of these three tasks, selecting $J_{3,1}$ can gain some utilities to the system but avoids the opportunity of gaining more utilities by running jobs of T_1 and T_2.

The idea of our improvement to the UPA and $Pseudo\ \alpha$heuristics is to consider the release time of future jobs. If the selected job can postpone some of future jobs while the utility value of the selected job is not much more than the sum of utilities of those future jobs, we can disregard the selected job and let the system to be idle. In this case we may miss some immediate utilities but we get an opportunity to the system to gain more utilities from the jobs that will arrive later. As a result our improvement is to disregard some selected jobs (that means to select a_{idel}) when future jobs can have more utilities. We consider two conditions for selecting a_{idle} instead of other actions: The workload of the system and the utility densities of the selected and future jobs. According to our experiments we use the following approach to formulize these two conditions. LetJ_{cur}be the selected job and J_{next} be the first next job that should be arrived to the ready queue and let t_{next} and p_{next} be its release time and its period. The system should usea_{idle} instead of the selected action if:

1- $t_{system} + ExpTime(J_{cur}) > t_{next}$, and
2- $\rho(t_{next}, t_{next} + p_{next}) > .8$, and
3- $\frac{U_{J_{cur}}(0)}{ExpTime(J_{cur})} < \frac{U_{J_{next}(0)}}{2\times[ExpTime(J_{next}) + t_{next} - t_{system}]}$

The first condition means that the running of the selected job can influence the start time of the first next job. If it doesn't hold, the selected job cannot affect any next jobs and we should not disregard the selected job.

The second condition shows that the workload of the system will be high during the running of the first next job and running the selected job may cause an overload to the system. The third condition considers utility density of the selected job (that is $\frac{U_{J_{cur}}(0)}{ExpTime(J_{cur})}$) and the utility density of the next job. For the second one, we should also notice that the system will be idle until the start of the next job (the system will be idle for $t_{next} - t_{system}$ units of time). Hence, we bias the utility density of the first next job to $\frac{U_{J_{next}(0)}}{ExpTime(J_{next}) + t_{next} - t_{system}}$.To be more conservative, utility density of the selected job should be less than half of the biased utility density of the first next job. Note that the second and third conditions are defined as heuristic conditions that are set from some experiments and there is not any guarantee for their correctness, but experimental results show that they are more useful than previous proposed heuristics. In Section 7 we consider other parameters for these two conditions.

VI. EXTENDING UA SCHEDULING FOR ENERGY HARVESTING SYSTEMS

In this section we introduce an extension of the UA scheduling problem for embedded systems that collect and store energy from their environment. These systems (that are called energy-harvesting systems)have some energy collector (such as solar panel) and energy storage units (battery or capacitor). Because of limited capacity of storage units in energy harvesting systems, the processor should be switched off in some times to let the energy storage unit to be recharged [31]. A common problem in many variants of energy harvesting systems is to optimize their performance or lifetime [42]. For example, because of limited lifetime of batteries and also different speed of processors (which means different levels of energy consumption) an optimal scheduling should consider a trade-off between the speed of the processor and

also the lifetime of storage units [29]. In this section we consider the UA scheduling problem of an embedded system with a battery as the storage unit. For the sake of simplicity we use a linear modelling for the battery (like [31]) and we suppose that the energy consumption of each executing job is one unit per time and the battery is recharged with the rate of two units per time when the system is idle. For this class of embedded systems, the scheduler is allowed to select a job if the current level of energy in the battery is not less than the needed energy for the WCET. In this case the scheduler should also consider the current energy level of the battery and decide to select one job from the ready queue or set the system to the idle mod for recharging the battery. We do not consider any limitation on the lifetime of the battery and it can be recharged to its maximum capacity without any limitation.

A. MDPs for Energy Harvesting Systems

We extend the MDP based approach for optimal UA scheduling of energy harvesting systems by considering an additional variable E for capturing the level of energy in the storage unit(battery).In this case every state of the MDP is of the form $s = (t_{system}, b_1, \dots, b_n, E)$ where the value of E is in the range $[0, E_{max}]$ where E_{max} is the maximum capacity of the battery. For this class of scheduling problem, transitions of the MDPs should also consider the level of energy in the battery. In each state s a probabilistic transition with an action a_i can be done if the value of E is at least equal to the WCET of the job of T_i.We use this condition because we suppose that the system cannot switch to the recharging mod before finishing the current job. Destination states of transitions can be defined like the definition of Section 4-2 and also by considering the value of E. With any action a_i related to the selection of a job J_i from the ready queue, a transition is defied from $s = (t_{system}, b_1, \dots, E)$ to $s'(t_{system} + t', b_1', \dots, E - t')$ with probability $D_i(t')$ if $b_i = true$ and $E \geq t'$. For the a_{idle} action, the transition from s will point to $s' = \big((t_{system} + 1) mod\ H, \dots, \min(E + 2, E_{max}\ )$ with probability 1.

B. Extending Heuristics for UA Scheduling of Energy Harvesting System

For extending previous heuristics one may need to consider the level of energy in the battery. According to the definition of EDF and greedy heuristics we don't need to apply any change to these heuristics and we only extend UPA and $pseudo\ \alpha$ heuristics for energy harvesting systems. For extending the idea of UPA for battery based systems we need to consider the level of battery in computations. Specially $\Delta_{i,j}$ should be defined as follows to consider the level of energy in the battery:

$$\Delta_{i,j}(t, E) = \sum_{d_1=l_i}^{w_i} D_i(d_1).[U_i\big(t\ mod\ p_i\ + d_1\big) + \sum_{d_2=l_j}^{\min(w_j, E-d_1)} D_j(d_2).U_j\big(t\ mod\ p_j\ + d_1 + d_2\big)]$$

–

$$\sum_{d_1=l_j}^{w_j} D_j(d_1).[U_j\big(t\ mod\ p_j\ + d_1\big) + \sum_{d_2=l_i}^{\min(w_i, E-d_2)} D_i(d_2).U_i\big(t\ mod\ p_i\ + d_1 + d_2\big)]$$

In this definition, running the first job will reduce the amount of the energy in the battery and only those runs of the second job are considered that can be done with the remaining level of energy in the battery (those jobs that their execution time is not more than $E - d_1$).

Although the definition of $\Delta_{i,j}$ for UA scheduling problems usually leads to good local policies [9], [14] but it is not much useful for UA scheduling of energy harvesting systems when the workload of the system is near 1 or more. In this case the system usually doesn't have enough energy to perform two consequence jobs. Instead, the system should be idle in appropriate points of time to increase the level of energy in the battery to perform more future jobs with higher utilities. For this purpose the scheduler should avoid the selected action and use a_{idle} if the workload of the system is relatively high ($\rho(t_{next}, t_{next} + p_{next}) > .8$) and at least one of the following conditions hold:

1- $\frac{U_{J_{cur}}(0)}{ExpTime(J_{cur})} < \frac{U_{J_{next}}(0)}{ExpTime(J_{next}) + t_{next} - t_{system}}$

2- $E < 0.4 \times E_{max}$ and $\frac{U_{J_{cur}}(0)}{ExpTime(J_{cur})} < AV\,G_{i=1}^{n} \frac{U_{J_i}(0)}{ExpTime(J_i)}$

where J_{next} is the first next job that its WCET is not more than $E + 2 \times (t_{next} - t_{system})$. The first condition holds when the utility density of the selected job is less than the biased utility density of the first next job. The second condition holds when the energy level of the battery is less than 40% of the maximum of energy level in the battery and the utility density of the selected job is less than the average of utility densities of tasks of the system. In the case of high workload and low energy level in the battery, the system should sacrifice some jobs with relatively low utility density in order to have more energy for doing other jobs with more utility densities.

VII. EXPERIMENTAL RESULTS

We simulate heuristics of Sections5 and 6 on several classes of UA scheduling problems with different workloads and different number of tasks. Our approach for defining case studies is the same as the approach of [4]. We use the algorithms of [4], [9] for implementing the UPA and Pseudo α heuristics and use $\alpha = 0$ in all cases. We mainly consider linear drop time utility functions while upper bound of each function is defined randomly in the range [2,32]. The value of utility function U_i is equal to the upper bound if the job finishes before a c_i time and this value decreases linearly to reach zero in its relative deadline τ_i.The period p_i of each task T_i is generated randomly in the range [100,2400] while the hyper period of every task set is $H = 2400$.

A. Comparing the Performance of Heuristics for Several Classes of Heavy Load Soft Real-time Systems

For the first set of simulations we consider 4sets of UA scheduling problems and for each set we consider 100 problem instances. Every problem of the first set contains 5 tasks while the second, the third, and the fourth sets contain 6,7 and 8 tasks for each problem, respectively. For each set, the lower bound l_i and upper bound w_i for the running time of the jobs of each T_i are generated to be in $l_i \in [\frac{.65\times p_i}{number\_of\_tasks}, \frac{.75\times p_i}{number\_of\_tasks}]$ and $w_i \in [\frac{1.25\times p_i}{number\_of\_tasks}, \frac{1.35\times p_i}{number\_of\_tasks}]$ .In this case the average workload of every system is near 1.The parameter c_i of the utility function is randomly generated with uniform distribution from the range $[w_i, p_i]$. The relative deadline τ_i is randomly generated from the range$[c_i, p_i]$. Figure 3 shows the average performance of 6 different heuristics. Each column shows the percentage of optimal UA value for every class of problems. The performance of each heuristic is computed by comparing the expected value of UA of the heuristic with the optimal value that is achieved from the MDP-based method.

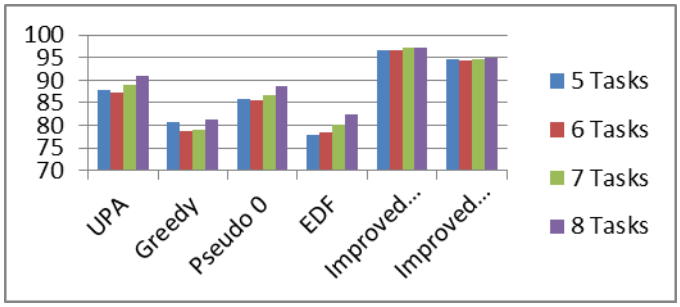


Fig. 3 Average performance of several heuristics

The performance of EDF and Greedy heuristics are less than others. The improvement of our method is about 8% in average for both UPA and $pseudo\ 0$ heuristics.It is more interesting that the performance of improved pseudo heuristic is about 5% more than the performance of standard UPA while its overhead is much less than the overhead of UPA. Note that the standard UPA performs better for the systems with more number of tasks but its overhead is an obstacle to use it in many real applications. The improved $pseudo\alpha$ heuristic on the other hand performs better than UPA for systems with more than 8 tasks.

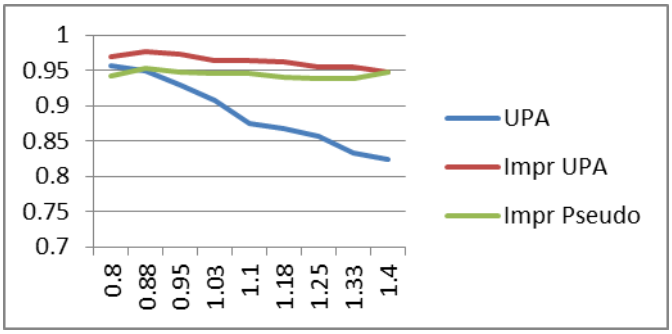
Table I shows for each heuristic, what fraction of the problem instances achieved at least 80%, 85%, 90% and 95% of optimal expected values. This table shows that for all problem instances, the improved UPA is at least 80% of optimal but about 24% of problem instances are less than 80% of optimal for the UPA heuristic. On the other hand the performance of Greedy and EDF is usually less than 95% of optimal while about 75% of instances are at least 95% optimal when the improved UPA is used.

B. Performance of Heuristics for Systems with Different Workload

Most of proposed heuristics for UA scheduling problems perform near optimal for the systems that rarely encounter overload scenarios [7].On the other hand the performance of some heuristics degrades when the workload of the system is high. We evaluate the impact of workload of the system on the performance of standard and improved UPA and also Improved Pseudo α heuristics.We consider several set of problems with different l_i and w_i parameters and use $\rho(0,H)$ as the workload criterion of every set of problems. Each set contains 50 problems with the hyper period $H = 1200$.Figure 4 shows the results of these experiments for a system with 6 tasks and a system with 8 tasks. For the systems with low workload, both versions of UPA perform near optimal but for systems with heavy load ($\rho(0,H) \geq .9$ in this example) improved UPA is usually more than 95% near to optimal but the performance of standard UPA is around 86% of optimal.The performance of $Pseudo$ is 94% near to optimal. An interesting result of these experiments is that the performance of UPA decreases when the workload of the system is more than one but the performance of improved UPA and improved Pseudoα is relatively constant.

TABLE I
Fraction of Optimal Vs Fraction of Experiments

Fraction of Optimal	Greedy	EDF	UPA	Pseudo α	Improved UPA	Improved Pseudoα
80%	51%	47%	79%	76%	100%	99%
85%	30%	23%	64%	58%	99%	95%
90%	18%	7%	46%	36%	98%	90%
95%	2%	1%	31%	12%	75%	54%



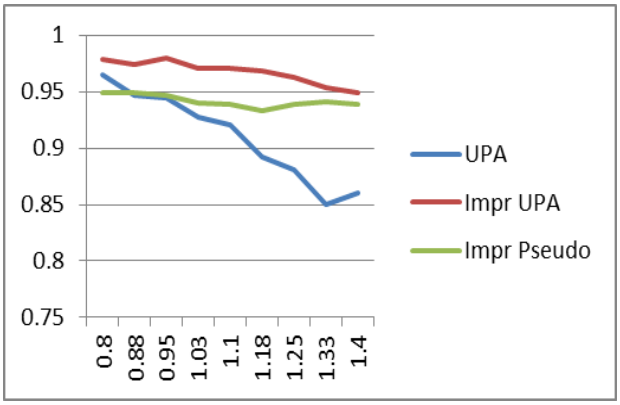


Fig. 4 Performances of 3 heuristics with different workloads for systems with 6 tasks (top) and 8 tasks (down)

C. Different Parameters for Improving UPA

In Section 5, we use 3 conditions that the scheduler should consider for selecting a_{idle}. We can use two parameters for the second and the third condition and suppose them as:

- $\rho(t_{next}, t_{next} + p_{next}) > \beta_1$
- $\frac{U_{J_{cur}}(0)}{ExpTime(J_{cur})} < \frac{U_{J_{next}}(0)}{\beta_2 \times [ExpTime(J_{next}) + t_{next} - t_{system}]}$

and consider the impact of different parameters on the performance of improved UPA. Fig 5 represents the impact of these two parameters on the performance of 50 problem instances with 6 tasks. According to these results, the impact of different values for these two parameters is about 1%. It is better to consider β_1 to be less than one and β_2 to be in the range of [1.5 , 2].

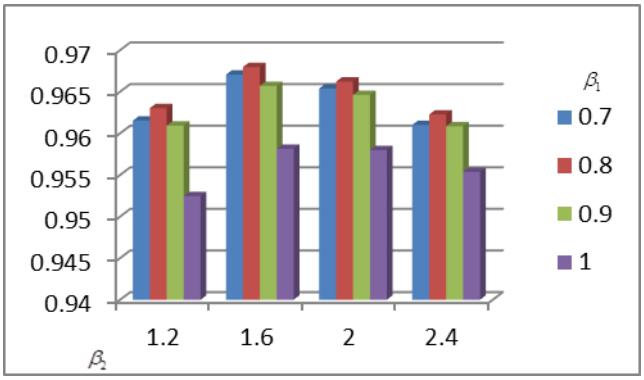


Fig. 5 Different parameters for conditions of improved UPA

D. Comparing Running Time of Heuristic Methods

The overhead of a heuristic scheduler can affect its performance and real-time embedded systems usually need to use schedulers with negligible running time. For better understanding of this overhead we compare the running time of each heuristic. We compute the average running time of each heuristic per state and the results are presented in Table II. We implemented these 6 heuristics in C++ and executed them on a corei3 Intel machine with 4 GB of main memory and all reported times are in millisecond. While the running time of greedy and EDF heuristics are less than .5 ms, the running time of Pseudoα is about 5 ms and the running time of the UPA heuristic is about 10 times more than the running time of Pseudoα. The features of an embedded system can determine which heuristic is more useful for a real system.

Table II. Running time per state for 6 heuristics

		Gree dy	ED F	Pseud oα	UPA	Improv ed Pseudo α	Improv ed UPA
6 Tasks	Average	0.307	0.273	4.842	45.07	4.913	47.31
	Max	0.433	0.338	10.892	98.356	11.023	101.492
	Min	0.196	0.143	2.174	22.312	2.235	23.518
5 Tasks	Average	0.291	0.142	3.607	38.041	3.648	40.31
	Max	0.385	0.226	7.113	81.717	7.242	84.487
	Min	0.089	0.113	1.093	6.107	1.131	6.33

E. Performance of heuristics for Energy harvesting Systems

We use a similar approach for simulating heuristics for energy harvesting systems. We consider two sets of 100 problem instances with hyper-period $H = 360$ with 5 tasks for the first set and 6 tasks for the second. Figure 5 shows the results of our experiments. According to these results, the performance of UPA for battery based systems is not so promising if we only modify the definition of $\Delta_{i,j}$ function and it needs more improvements. In this case the performance of EDF is also about 80% of optimal. On the other hand if we apply the modification of Section 6-2 the performance of improved UPA and improved Pseudo is more than other heuristics. Yet the performance of these two heuristics is about 90% of optimal and more improvement may be needed.

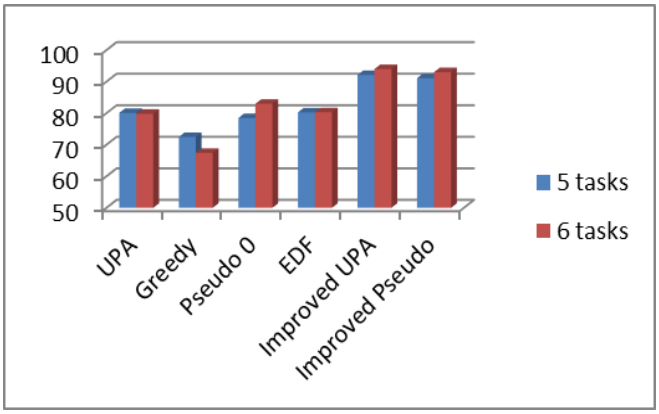


Fig. 5 Heuristics for Battery based UA scheduling problems

VIII. CONCLUSION AND FUTURE WORKS

In this paper we propose an improvement to the UPA and Pseudoα heuristics for stochastic UA scheduling problems. Experimental results show that our method improves the performance of previous heuristics to 95% of optimal. We also introduce an extension to the problem by consider battery based systems. In this case we also propose a modification for the UPA and Pseudoα heuristics. Some problems remain for future works. The impact of our improved heuristics for other time utility functions should be analysed. More realistic models of energy harvesting embedded systems can be considered and more improvements can be developed. Other approaches (like Timed Automata) can be used for modelling this problem.

REFERENCES

[1] Jensen, E. D., Locke, C. D., &Tokuda, H. (1985, December). A Time-Driven Scheduling Model for Real-Time Operating Systems.In RTSS (Vol. 85, pp. 112-122).

[2] Li, Peng, BinoyRavindran, and E. Douglas Jensen. "Utility Accrual Real-time Scheduling with Probabilistically-Assured Timeliness Performance." PARTES Workshop, ACM EMSOFT. 2004.

[3] Kluge, Florian, et al. "History-cognisant time-utility-functions for scheduling overloaded real-time control systems." Proceedings of 7th Junior Researcher Workshop on Real-Time Computing (JRWRTC). 2013.

[4] Tidwell, Terry. "Utility-aware scheduling of stochastic real-time systems." (2011).

[5] Li, Peng, et al. "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints." IEEE Transactions on Computers 55.4 (2006): 454-469.

[6] Wu, Haisang, BinoyRavindran, and E. Douglas Jensen. "Utility accrual real-time scheduling under the unimodal arbitrary arrival model with energy bounds." IEEE Transactions on Computers 56.10 (2007): 1358-1371.

[7] Tidwell, Terry, et al. "Optimizing expected time utility in cyber-physical systems schedulers." Real-Time Systems Symposium (RTSS), 2010 IEEE 31st. IEEE, 2010.

[8] Li, Peng, et al. "Stochastic, Utility Accrual Real-Time Scheduling with Task-Level and System-Level Timeliness Assurances." ISORC.Vol. 5. 2005.

[9] Tidwell, Terry, et al. "Scalable utility aware scheduling heuristics for real-time tasks with stochastic non-preemptive execution intervals." 2011 23rd Euromicro Conference on Real-Time Systems.IEEE, 2011.

[10] Abeni, L., et al. "Soft real-time systems: predictability vs. efficiency." Massachusetts institute of technology-2005 (2005).

[11] Stankovic, John A., et al. "Implications of classical scheduling results for real-time systems." Computer 28.6 (1995): 16-25.

[12] Clark, Raymond Keith. Scheduling dependent real-time activities.Diss. Carnegie Mellon University, 1990.

[13] Guerra, Raphael, and Gerhard Fohler. "A gravitational task model with arbitrary anchor points for target sensitive real-time applications." Real-Time Systems 43.1 (2009): 93-115.

[14] Wang, Jinggang, and BinoyRavindran. "Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis." IEEE Transactions on Parallel and Distributed Systems 15.2 (2004): 119-133.

[15] Li, Peng. Utility accrual real-time scheduling: Models and algorithms. Diss. Virginia Polytechnic Institute and State University, 2004.

[16] Yu, Yue, et al. "Profit and penalty aware (pp-aware) scheduling for tasks with variable task execution time." Proceedings of the 2010 ACM Symposium on Applied Computing.ACM, 2010.

[17] Li, Shuhui, et al. "Profit and penalty aware scheduling for real-time online services." IEEE Transactions on industrial informatics 8.1 (2012): 78-89.

[18] Liu, Shuo, Gang Quan, and ShangpingRen. "On-Line Real-Time Service-Oriented Task Scheduling Using TUF."ISRN Software Engineering 2012 (2012).

[19] Li, Peng. Utility accrual real-time scheduling: Models and algorithms. Diss. Virginia Polytechnic Institute and State University, 2004.

[20] Manolache, Sorin, PetruEles, and ZeboPeng. "Schedulability analysis of applications with stochastic task execution times." ACM Transactions on Embedded Computing Systems (TECS) 3.4 (2004): 706-735.

[21] Fehnker, Ansgar. Scheduling a steel plant with timed automata.Computing Science Institute Nijmegen, Faculty of Mathematics and Informatics, Catholic University of Nijmegen, 1999.

[22] Abdeddaïm, Yasmina, AbdelkarimKerbaa, and OdedMaler. "Task graph scheduling using timed automata." Parallel and Distributed Processing Symposium, 2003. Proceedings.International.IEEE, 2003.

[23] Abdeddaïm, Yasmina, and Damien Masson. "Scheduling self-suspending periodic real-time tasks using model checking."RTSS 2011 WiP. 2011.
[24] Bouyer, Patricia, et al. "Quantitative analysis of real-time systems using priced timed automata." Communications of the ACM 54.9 (2011): 78-87.
[25] Krčál, Pavel, and Wang Yi. "Decidable and undecidable problems in schedulability analysis using timed automata."International Conference on Tools and Algorithms for the Construction and Analysis of Systems.Springer Berlin Heidelberg, 2004.
[26] Abdeddaı, Yasmina, Eugene Asarin, and OdedMaler. "Scheduling with timed automata." Theoretical Computer Science 354.2 (2006): 272-300.
[27] Kempf, Jean-Francois, Marius Bozga, and OdedMaler. "As soon as probable: Optimal scheduling under stochastic uncertainty." International Conference on Tools and Algorithms for the Construction and Analysis of Systems.Springer Berlin Heidelberg, 2013.
[28] Mader, Angelika, et al. "Synthesis and stochastic assessment of cost-optimal schedules." International journal on software tools for technology transfer 12.5 (2010): 305-318.
[29] Wu, Haisang, BinoyRavindran, and E. Douglas Jensen. "Energy-efficient, utility accrual real-time scheduling under the unimodal arbitrary arrival model."Proceedings of the conference on Design, Automation and Test in Europe-Volume 1.IEEE Computer Society, 2005.
[30] Niu, Jianwei, et al. "Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems." IEEE Transactions on Parallel and Distributed Systems 25.8 (2014): 2043-2052.
[31] Real-Time Scheduling of Energy Harvesting Embedded Systems with Timed Automata
[32] Baums, Aldis. "Indicators of the real time of a mobile autonomous robot." Automatic Control and Computer Sciences 46.6 (2012): 261-267.
[33] Moon, Hyun Jin, Yun Chi, and HakanHacigümüş. "Performance evaluation of scheduling algorithms for database services with soft and hard SLAs." Proceedings of the second international workshop on Data intensive computing in the clouds.ACM, 2011.
[34] Santhosh, R., and T. Ravichandran. "Pre-emptive scheduling of on-line real time services with task migration for cloud computing." Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on.IEEE, 2013.
[35] Liu, Shuo, Gang Quan, and ShangpingRen. "On-line scheduling of real-time services for cloud computing." 2010 6th World Congress on Services.IEEE, 2010.
[36] Michelusi, Nicolo, and Michele Zorzi. "Optimal adaptive random multiaccess in energy harvesting wireless sensor networks." IEEE Transactions on Communications 63.4 (2015): 1355-1372.
[37] Zhang, Yongmin, et al. "Distributed sampling rate control for rechargeable sensor nodes with limited battery capacity." IEEE Transactions on Wireless Communications 12.6 (2013): 3096-3106.
[38] Puterman, Martin L. Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, 2014.
[39] Dai, Peng, Daniel S. Weld, and Judy Goldsmith. "Topological value iteration algorithms." Journal of Artificial Intelligence Research 42 (2011): 181-209.
[40] Bellman, Richard. "Dynamic programming and Lagrange multipliers." Proceedings of the National Academy of Sciences 42.10 (1956): 767-769.
[41] Liu, Chung Laung, and James W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment." Journal of the ACM (JACM) 20.1 (1973): 46-61.
[42] Chetto, Maryline. "Optimal scheduling for real-time jobs in energy harvesting computing systems." IEEE Transactions on Emerging Topics in Computing 2.2 (2014): 122-133.