

Utility Accrual Real-Time Scheduling Under Variable Cost Functions *

Haisang Wu* Umut Balli* Binoy Ravindran* E. Douglas Jensen†

**ECE Dept., Virginia Tech*

†*The MITRE Corporation*

Blacksburg, VA 24061, USA

Bedford, MA 01730, USA

{hswu02, uballi, binoy}@vt.edu

jensen@mitre.org

Abstract

We present a utility accrual real-time scheduling algorithm called VCUA, for tasks whose execution times are functions of their starting times. We model such variable execution times with *variable cost functions* (or VCF). The algorithm considers application activities that are subject to time/utility function time constraints, VCFs, and the multi-criteria scheduling objective of assuring that the maximum interval between any two consecutive, successful completion of jobs *of a task* must not exceed a specified bound, and maximizing the system's total utility. Since the scheduling problem is intractable, VCUA off-line selects tasks based on their potential utility density, and dynamically promotes jobs to accrue more utility, in polynomial-time. We establish that VCUA achieves optimal timeliness during under-loads, and identify the conditions under which timeliness assurances hold. Our simulation experiments illustrate VCUA's superiority.

1 Introduction

Many emerging real-time embedded systems such as robotic systems in the space domain (e.g., NASA's Mars Rover [4]) and control systems in the defense domain (e.g., phased array radars [8] and battle management systems [7]) are subject to time constraints that are "soft" (besides those that are hard) in the sense that completing an activity at any time will result in some (positive or negative) utility to the system, and that utility depends on the activity's completion time. These soft time constraints are subject to optimality criteria such as completing all time-constrained activities as close as possible to their *optimal* completion times—so as to yield maximal collective utility. The optimality of the soft time constraints is generally as mission- and safety-critical as that of the hard time constraints.

*This work was supported by the US Office of Naval Research under Grant N00014-00-1-0549 and The MITRE Corporation under Grant 52917.

Jensen’s time/utility functions [11] (or TUFs) allow the semantics of soft time constraints to be precisely specified. A TUF, which generalizes the deadline constraint, specifies the utility to the system resulting from the completion of an activity as a function of its completion time. A TUF’s utility values are derived from application-level quality of service metrics. Figure 1 shows example time constraints from real applications specified using TUFs. Figures 1(a)–1(c) show time constraints of two applications in the defense domain [2, 17]. The classical deadline is a binary-valued, downward “step” shaped TUF; 1(d) shows an example.

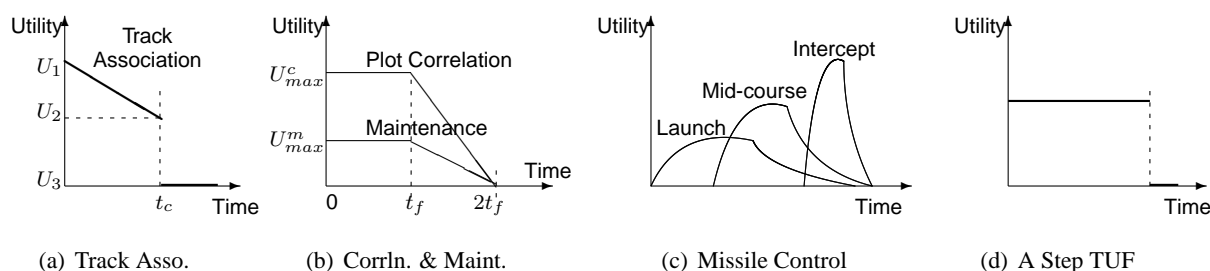


Figure 1: Example TUF Time Constraints from AWACS [2] (a) and Coastal Air Defense [17] (b-c), and a Step TUF (d).

When activity time constraints are expressed with TUFs, the timeliness optimality criteria are typically based on accrued activity utility—e.g., maximizing sum of the activities’ attained utilities or assuring satisfaction of lower bounds on activities’ maximal utilities. Such criteria are called *Utility Accrual* (or UA) criteria, and sequencing (scheduling, dispatching) algorithms that consider UA criteria are called UA sequencing algorithms.

UA criteria facilitate adaptivity during overloads, when completing activities that are more important than those which are more urgent is often desirable. During overloads, UA algorithms that maximize summed utility typically favor activities that are more important (since more utility can be attained from them) than those which are more urgent. Thus, the optimal timeliness behavior of deadline scheduling during under-loads [9] is a special case of UA scheduling.

1.1 Variable Cost Scheduling

In this paper, we focus on variable cost scheduling. By variable cost scheduling we mean scheduling activities that have durations (e.g., tasks with execution times) which vary while being performed—e.g., depending on when they begin, or on how long they have been running, or on other factors. In this context, cost means the duration of the activity—a term that comes from one of the interesting and important applications for such scheduling. The variable cost is specified by a cost function. Thus, even if there were no new activity arrivals, the load to be scheduled changes while the activities are being performed.

Past efforts on deadline-based and utility accrual real-time scheduling do not consider variable cost schedul-

ing. For example, UA scheduling algorithms presented in the literature [1, 3, 12, 13, 16, 19, 20] have task execution times as constant values (or constant mean values), which do not vary while tasks are being performed. The *imprecise computations* [15] and *IRIS (Increasing Reward with Increasing Service)* [6] models propose optional parts in addition to the mandatory parts of task execution times. But these concepts are different from UA and variable cost scheduling, because (1) in these models the longer the optional part executes, the higher the reward, while in UA scheduling the utility (reward) can only be accrued by an activity when its completed, and the utility value is decided by the completion time; and (2) there are no optional parts in variable cost scheduling—task execution times only contain the mandatory parts and they may vary while the tasks are being performed.

Thus, no past efforts studied the intersection of UA scheduling and variable cost scheduling. In this paper, we precisely study this problem. We consider repeatedly occurring, real-time application activities whose time constraints are specified using TUFs. The activities execution times are described with cost functions, which may vary as the activities are being performed. For such an application model, we consider a two-fold scheduling criterion: (1) assure that the maximum interval between any two consecutive, successful completions of jobs *of a task* must not exceed a specified bound; and (2) maximize the system’s summed utility.

This problem is \mathcal{NP} -hard. We present a polynomial-time heuristic algorithm for the problem, called *Variable Cost Utility Accrual Algorithm* (or VCUA). Further, we prove several timeliness properties of the algorithm including timeliness optimality during under-loads, and identify the conditions under which timeliness assurances hold. Finally, our simulation studies confirm the effectiveness and superior performance of VCUA.

Thus, the contribution of the paper is the VCUA algorithm. To the best of our knowledge, we are not aware of any other efforts that solve the problem solved by VCUA.

The rest of the paper is organized as follows: In Section 2, we describe the motivating applications for variable cost scheduling; In Section 3, we outline our activity and utility models, and state the scheduling criterion. We present VCUA in Section 4 and establish the algorithm’s timeliness properties in Section 5. The experimental measurements are reported in Section 6. Finally, we conclude the paper and identify future work in Section 7.

2 Motivating Applications

The application context of interest to us in this paper for variable cost scheduling is a type of very complex air-to-air tracking problem for which scheduling algorithms and performance assurances have not been publicly available. For the purpose of the work summarized in this initial paper, we simplify and omit some character-

istics of the tracking problem to expedite our creation of an initial plausible scheduling approach that can be generalized in subsequent work, and to avoid U.S. DoD classification issues.

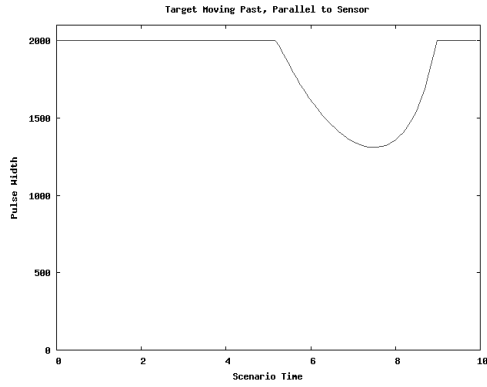
This type of tracking problem uses an electronically steerable phased array radar on a sensor aircraft. The problem notionally consists of three component tasks: searching a segment of the airspace designated by another entity; identifying any targets found in that airspace segment; and maintaining the track for each of those targets until some deadline time. Those three tasks for a given target nominally occur in that order, but identification can occur almost anytime while tracking. The tasks may be preemptive (here we assume they are not). The three tasks for a given target can be interleaved with the three tasks for other targets.

For each of the three tasks, the radar must illuminate the target with pulses called dwells. Preempting the dwell of an identification task is very expensive because it is lengthy and requires the task be restarted from the beginning, so we presume that task is not preemptive. For the search and tracking tasks, the dwells occur at a revisit rate that is defined by the interval between two successive dwells—not necessarily periodically. The revisit rate for any particular target must be maintained for a long enough time to maintain acceptable values for certain application-level quality of service (AQoS) metrics. One critical such AQoS metric is track quality, which is a measure of how well known is a given target’s location and motion. Achieving any particular track quality value imposes a lower bound on the revisit rate of the target being tracked. Optimal and minimum revisit rates are defined by the probability of detecting the target with the next dwell—failure to meet a minimum revisit rate implies increased chance of missing the target on the next dwell. Another metric of interest is efficiency of the radar utilization—e.g., the track quality that can be maintained for a given percentage utilization of the radar. Perhaps contrary to intuition, maximizing radar utilization is neither necessary nor sufficient to attain various AQoS metrics; intentional radar idle time can result in improved AQoS metrics.

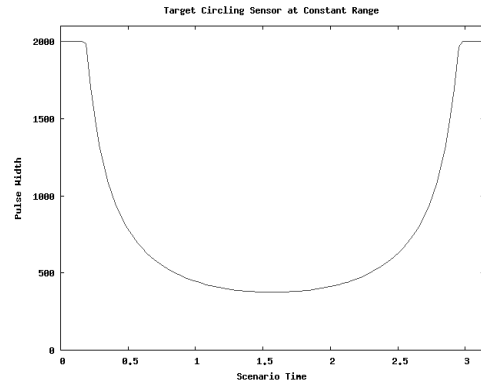
In this application, activity cost varies with many factors, including the geometry of the situation. The time to scan a segment of airspace, and the length of the dwells, depend on the relative positions of the radar platform, and the scanned airspace and the targets in that space. Depending on the relative motions of the radar platform and the target, it may be better to either procrastinate dwells (intentionally insert idle time in the radar schedule) or perform dwells early; here we assume that the scheduler does neither.

The cost function for each task varies with each target’s range and look angle (azimuth off the sensor’s boresight)—i.e., having the form $f(r, \theta)$. The cost function is derived from the particular tracking problem and an equation called the radar equation [18]. The radar equation relates the measured energy received to the geometry of the target, the sensor, and the emitted energy.

Two examples of cost functions are shown in Figure 2. Figure 2(a) shows the cost function for a target flying at a higher altitude and faster velocity than the radar platform; Figure 2(b) shows the cost function for a



(a) Example Cost Function 1



(b) Example Cost Function 2

Figure 2: Example Cost Functions for (a) a Target Flying at a Higher Altitude and Faster Velocity than the Radar Platform, and (b) a Target Circling the Radar Platform at a Constant Range

a target circling the radar platform at a constant range. The cost functions of real applications can be increasing, decreasing, or strictly convex shaped. In this paper, we make the simplifying assumption that cost functions are non-decreasing for our initial study of this problem.

3 Models and Objective

3.1 System and Task Model

We consider a non-preemptive system which consists of a set of independent periodic tasks, denoted as $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$. Each task T_i contains a collection of instances. The period of a task T_i is denoted as P_i . Each task has a begin time and an end time between which execution of all jobs of the task must be completed.

An instance of a task is called a *job*, and we refer to the j^{th} job of task T_i , which is also the j^{th} invocation of T_i , as $J_{i,j}$. The basic scheduling entity we consider is the job abstraction. Thus, we use J to denote a job without being task specific, as seen by the scheduler at any scheduling event; J_k can be used to represent a job in the job scheduling queue.

3.2 Timeliness Model

A job's time constraint is specified using a TUF. Jobs of a task have the same TUF. We use $U_i(\cdot)$ to denote task T_i 's TUF, and use $U_{i,j}(\cdot)$ to denote the TUF of T_i 's j^{th} job. Without being task specific, $J_k.U$ means the TUF of a job J_k ; completion of J_k at a time t will yield a utility $J_k.U(t)$.

TUFs can be classified into unimodal and multimodal functions. Unimodal TUFs are those for which any decrease in utility cannot be followed by an increase. Examples are shown in Figure 1. TUFs which are not

unimodal are multimodal. In this paper, we restrict our focus to *non-increasing*, unimodal TUFs i.e., those TUFs for which utility never increases as time advances. Figures 1(a), 1(b), and 1(d) show examples.

Each TUF $U_{i,j}, i \in \{1, \dots, n\}$ has an initial time $I_{i,j}$ and a termination time $X_{i,j}$. Initial and termination times are the earliest and the latest times for which the TUF is defined, respectively. We assume that $I_{i,j}$ is equal to the arrival time of $J_{i,j}$, and $X_{i,j} - I_{i,j}$ is equal to the period P_i of the task T_i . If a job's termination time is reached and its execution has not been completed, an exception is raised. Normally, this exception will cause the job's abortion and execution of exception handlers.

3.3 Task Execution Time Model

After a job is released, the job's execution time varies with time. Thus, we define a *variable cost function* (or VCF) for each job, which describes the job execution time as a function of its starting time. Jobs of a task have the same VCFs, so a VCF is also defined for a task. We use $C_i(\cdot)$ to denote task T_i 's VCF, and use $C_{i,j}(\cdot)$ to denote the VCF of T_i 's j th job.

For a job $J_{i,j}$, the x -axis of its VCF is the absolute time according to the job's arrival time; the y -axis represents its execution time $C_{i,j}(t)$, and the origin shows the execution time of $J_{i,j}$ when it is just released.

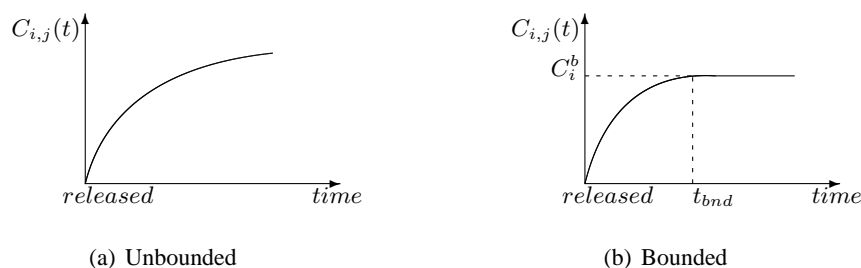


Figure 3: Variable Cost Functions of Job $J_{i,j}$

In this paper, we consider non-decreasing VCFs. Thus, as time proceeds, the execution time of a job never decreases. We categorize VCFs into two types: unbounded and bounded. Figure 3 shows these two types. Figure 3(a) shows an unbounded VCF, where $C_{i,j}(t)$ increases with $time$ indefinitely. Figure 3(b) shows a bounded VCF, where $C_{i,j}(t)$ is bounded by C_i^b , so that after t_{bnd} , $C_{i,j}(t)$ is equal to C_i^b .

Without being task specific, $J_k.VCF$ or $J_k.C$ means the VCF of a job J_k ; the execution time of J_k at a time t_{cur} will be $J_k.C(t_{cur}) = J_k.VCF(t_{cur})$. Hereafter, in discussion of TUF and VCF, we interchangeably use the terms *task* and *job* if no confusion is raised.

3.4 Scheduling Objective

A successful completion of a job means that the job has met its termination time. With this definition, we consider a two-fold scheduling criterion: (1) assure that the maximum interval between any two consecutive, successful completions of jobs *of a task* must not exceed a specified bound; and (2) maximize the system's summed utility.

Note that with VCFs, it is difficult to statically calculate the system load, since it dynamically varies with time. Even a constant load at the task arrivals—one that is an under-load—gradually becomes a heavier load, and may eventually become an overload even without new tasks arriving, due to increasing VCFs. Therefore, if the dynamically system load is too high such that scheduling objective (1) cannot be satisfied for each task, some tasks may be dropped. In such case, tasks that are not dropped are still subject to the two scheduling objectives. We propose VCUA to solve this problem, and its details are described in Section 4.

4 The VCUA Algorithm

4.1 Algorithm Rationale

The potential utility that can be accrued by executing a job defines a measure of its “return on investment.” Because of the unpredictability of future events, scheduling events that may happen later such as job completion and new job arrivals cannot be considered at the time when the scheduler is invoked. Thus, a reasonable heuristic is the “greedy” strategy, which means selecting as many “high return” jobs into the schedule as early as possible. This will increase the likelihood of maximizing the aggregate utility.

The metric used by VCUA to determine the return on investment for a job is called the *Potential Utility Density* (or PUD), which was originally developed in [3]. The PUD of a job measures the amount of utility that can be accrued per unit time by executing the job.

To compute $J_{i,j}$'s PUD at current time t_{cur} , VCUA considers $J_{i,j}$'s expected completion time, which is denoted as $t_{cur} + C_{i,j}(t_{cur})$, and the expected utility by executing $J_{i,j}$. PUD of $J_{i,j}$ is then computed as:

$$\frac{U_{i,j}(t_{cur} + C_{i,j}(t_{cur}))}{C_{i,j}(t_{cur})}.$$

4.2 Static Selection of VCUA

We assume that if a job cannot complete before its termination time even though it is scheduled immediately, it is infeasible and can be safely aborted. This process is called feasibility check. Details of feasibility check will be described in Section 4.3. With feasibility check, by solving the inequality $C_{i,j}(t) + t \leq X_{i,j}$, we can derive the latest possible starting time $t_{i,j}^b$ of job $J_{i,j}$, such that $C_{i,j}(t_{i,j}^b) + t_{i,j}^b = X_{i,j}$. Apparently, $C_{i,j}(t_{i,j}^b)$

corresponds to the maximum possible execution time of $J_{i,j}$, and $C_i(t_i^b)$ describes this parameter at the task level.

Therefore, although a job's execution time changes with its starting time, it is possible for us to derive a system load bound $load_b = \sum_{i=1}^n \frac{C_i(t_i^b)}{P_i}$, which will never be exceeded by the system's dynamic load. If no VCF is defined for each task, then a task's execution time is constant and $load_b$ here is the same as the system utilization definition in [14].

Since the system dynamic load may gradually increase even without new task arrivals, the task instances to be executed must be carefully selected in order to accrue more utility. Such selection process is guided by the performance metric PUD. To this aim, we use a job selection flag in which each task instance (job) is labeled as *skipped* or *selected*. The selection process considers the parameters of the task set such as VCFs and TUFs. We associate with each job $J_{i,j}$ a label $SEL_{i,j}$, where $SEL_{i,j} = \textit{skipped}$ indicates that the job is skipped and $SEL_{i,j} = \textit{selected}$ indicates that it is selected for execution. At run-time, only jobs whose labels are set to *selected* are dispatched. Thus, the problem becomes choosing the job labels for our scheduling performance objective.

We perform the labeling of jobs in both a static and dynamic fashion, based on the workload information used by the scheduler. In the off-line (static) part of VCUA, we select task instances before the application starts. Initially, all tasks in \mathbf{T} are labeled as *skipped*, i.e., $SEL_{i,j} = \textit{skipped}, \forall i \in 1, \dots, n, \forall j$. At $t_{cur} = 0$, we calculate PUD of each task, which in value is also the PUD of each task's first job, i.e., $PUD_i = \frac{U_{i,1}(C_{i,1}(0))}{C_{i,1}(0)}$. We also calculate the maximum possible execution time of each task $C_i(t_i^b)$, and then choose sub task set \mathbf{T}' . \mathbf{T}' consists of n' tasks with the largest PUDs, such that $load'_b = \sum_{i=1}^{n'} \frac{C_i(t_i^b)}{P_i} \leq 1$, and n' is the maximum possible tasks to be selected. To derive the sub task set \mathbf{T}' , every task in \mathbf{T} must be considered in an order of non-increasing PUD. Note that if $load_b \leq 1$, then $n' = n$. Thus, we favor tasks with larger PUDs, and label the n' tasks in \mathbf{T}' as *selected* i.e., $SEL_{i,j} = \textit{selected}, \forall i \in 1', \dots, n', \forall j$.

4.3 Dynamic Utility Accrual Scheduling

After the initial static steps, VCUA reserves the maximum sub task set consisting of the highest PUD tasks, whose dynamic load will not cause the system overload. VCUA then adopts the non-preemptive earliest termination time first (or NP-EXF) scheduling policy which is known to be optimal from the feasibility point of view [10, 14].

In the scheduling process, the selection scheme dynamically selects additional (optional) jobs to be executed for utility accrual, favoring jobs with higher PUDs. Such additional selection is performed at each scheduling event.

Since tasks are not preemptive, the scheduling events of VCUA include: (1) the arrival of a job and the expiration of a time constraint such as the arrival of a TUF's termination time, when the CPU is idle; and (2) the completion of a job. We define the following variables and auxiliary functions for VCUA:

- $\mathcal{J}_r = \{J_1, J_2, \dots, J_m\}$ is the current unscheduled job set; σ is the ordered output schedule. $J_k \in \mathcal{J}_r$ is a job. $J_k.X$ is its termination time, and $J_k.SEL$ is the job selection flag.
- $\text{findSEL}(\sigma)$ returns the first job in σ whose selection flag $SEL = \text{selected}$.
- $\text{headOf}(\sigma)$ returns the first job in σ .
- $\text{sortByPUD}(\sigma)$ returns a schedule by non-increasing PUD. If two or more jobs have the same PUDs, then the job(s) with the largest execution time should appear before any others with the same PUDs.
- $\text{insert}(J, \sigma, I)$ inserts J in the ordered list σ at the position indicated by index I ; if there are already entries in σ at the index I , J is inserted after them.
- $\text{feasible}(\sigma)$ returns a boolean value indicating schedule σ 's feasibility. For σ to be feasible, the predicted completion time of each job in σ must never exceed its termination time.

```

1: input   :  $\mathbf{T} = \{T_1, \dots, T_n\}, \mathcal{J}_r = \{J_1, \dots, J_m\}$ 
2: output  : selected job  $J_{exe}$ 
3: Initialization:  $t := t_{cur}, \sigma := \emptyset;$ 
4: for  $\forall J_k \in \mathcal{J}_r$  do
5:   if  $\text{feasible}(J_k) = \text{false}$  then  $\text{abort}(J_k);$ 
6:   else  $J_k.PUD := \frac{J_k.U - t + J_k.C(t)}{J_k.C(t)};$ 
7:  $\sigma_{tmp} := \text{sortByPUD}(\mathcal{J}_r);$ 
8: for  $\forall J_k \in \sigma_{tmp}$  from head to tail do
9:   if  $J_k.PUD \geq 0$  then
10:    copy  $\sigma$  into  $\sigma_{tent}$ :  $\sigma_{tent} := \sigma;$ 
11:     $\text{insert}(J_k, \sigma_{tent}, J_k.X);$ 
12:    if  $\text{feasible}(\sigma_{tent})$  then  $\sigma := \sigma_{tent};$ 
13:   else break;
14:  $J_{exe} := \text{findSEL}(\sigma);$ 
15: if  $J_{exe} = \emptyset$  then
16:    $J_{exe} := \text{headOf}(\sigma);$ 
17:    $J_{exe}.SEL := \text{selected};$ 
18: return  $J_{exe}$ 

```

Algorithm 1: VCUA: Dynamic Part Description

A high level description of VCUA is shown in Algorithm 1. At the beginning of each scheduling event, when VCUA is invoked at time t_{cur} , the algorithm first checks the feasibility of all the jobs in the current ready queue. If a job is infeasible, then it can be safely aborted (line 5). Otherwise, its PUD is calculated (line 6).

The jobs are then sorted by their PUDs (line 7). In each step of the **for** loop from line 8 to 13, the job with the largest PUD is inserted into σ , if it can produce a non-negative PUD and keep the schedule after insertion

feasible. Thus, σ is a feasible schedule sorted by the jobs' termination times, in a non-decreasing order.

From line 14 to line 18, VCUA finds and returns job J_{exe} to execute. At line 14, it tries to find the first job in σ with $SEL = selected$. If $findSEL()$ returns an empty result, which means $\sigma \in \mathbf{T} - \mathbf{T}'$, then the first job of σ is selected. Note that at run-time, only jobs whose labels are set to *selected* are dispatched, so we need to set $J_{exe}.SEL = selected$ at line 17. This process is called *job promotion*

Job promotion is the result from line 8 to line 18, and it occurs when there is excess CPU bandwidth due to some jobs' early completion. The `for` loop from line 8 to 13 implies that we favor tasks in \mathbf{T}' with higher PUDs, and $feasible(\sigma_{tent})$ in line 12 always returns true if $J_k \in \mathbf{T}'$, since $load'_b \leq 1$. When σ in line 14 contains both jobs from $\mathbf{T} - \mathbf{T}'$ and \mathbf{T}' , we do not perform job promotion. Only if at some scheduling event, σ exclusively contains jobs from $\mathbf{T} - \mathbf{T}'$, the `if` loop of line 15–17 is executed, and job promotion occurs.

5 Timeliness Properties of VCUA

We consider timeliness properties of VCUA, and compare it with a number of well-known algorithms. Specifically, we consider the following two conditions: (1) a set of independent periodic tasks without preemption; and (2) there are sufficient processor cycles for meeting all task termination times—i.e., there is no overload, and $load_b \leq 1$.

Theorem 1 *Under conditions (1) and (2), without inserted idle time [10], a schedule produced by non-preemptive EDF [9] is also produced by VCUA, yielding equal total utilities. Not coincidentally, this is simply a termination time ordered schedule.*

Proof We prove this by examining Algorithms 1. For periodic tasks during non-overload situations, σ from line 12 of Algorithm 1 is termination time ordered, due to the properties of the procedure `insert()`. The termination time that we consider is analogous to a deadline in [9]. As proved in [9, 10, 14], a deadline-ordered schedule is optimal (with respect to meeting all deadlines) for non-preemptive task sets when there are no overloads and no inserted idle time. Thus, σ yields the same total utility as non-preemptive EDF. \square

Some important corollaries about VCUA's timeliness behavior during non-overload situations can be deduced from EDF's optimality [5, 10].

Corollary 2 *Under conditions (1) and (2), without inserted idle time [10], VCUA always meets all task termination times.*

Corollary 3 *Under conditions (1) and (2), without inserted idle time [10], VCUA minimizes the maximum lateness.*

With the previous theorems and corollaries, we derive the property of VCUA in terms of the scheduling objective.

Theorem 4 *Under conditions (1) and (2), without inserted idle time [10], VCUA assures that the maximum interval between any two consecutive, successful completions of jobs of a task does not exceed the length of twice task period.*

Proof From Corollary 2, under conditions (1) and (2), without inserted idle time, VCUA can meet all task termination times. This ensures that for one task T_i , inside each of its period P_i , there is a successful completion of a job. We then consider two consecutive periods for task T_i . Due to T_i 's VCF, the largest interval between successful completions of jobs $J_{i,j}$ and $J_{i,j+1}$ happens when $J_{i,j}$ has the shortest possible execution time and completes right after it is released, and $J_{i,j+1}$ has the longest possible execution and completes right before its termination time. This largest interval is less than the length of $2P_i$. Therefore, for any task, the interval between two consecutive, successful completions of its jobs does not exceed the length of twice task period. \square

In Corollary 5, we also derive the above theorem' counterpart for the interval between two consecutive, successful completions of jobs of all tasks.

Corollary 5 *Under conditions (1) and (2), without inserted idle time [10], VCUA assures that the maximum interval between any two consecutive, successful completions of jobs of all tasks does not exceed the length of twice minimum task period.*

During system overload $load_b > 1$, VCUA dynamically performs job promotion to select additional jobs from $\mathbf{T} - \mathbf{T}'$ to execute for maximizing utility. Under certain conditions, even with promotion, the sub task set \mathbf{T}' with $load'_b \leq 1$ still conforms to Theorem 4. We prove this in Theorem 6.

Theorem 6 *Without inserted idle time, during system overload $load_b > 1$, if promoted jobs can complete before the next job arrival, VCUA assures that the maximum interval between any two consecutive, successful completions of jobs of a task in \mathbf{T}' does not exceed the length of twice task period.*

Proof We prove this by examining Algorithms 1. Since tasks in \mathbf{T}' have the highest PUDs and $load'_b \leq 1$, any instances of these tasks will be selected first in the `for` loop starting from line 8, and these instances will always be inserted into σ in line 12. This, jobs in \mathbf{T}' will always appear in the output queue σ even during system overloads.

We then analyze job promotion by analyzing σ when $\sigma \neq \emptyset$.

Case 1: some jobs in σ are labeled as *selected*, and the others are labeled as *skipped*.

It is easy to see that after line 14, J_{exe} can be found, and VCUA goes to line 18. No job promotion is performed.

Case 2: all jobs in σ are labeled as *skipped*.

In this case, $\text{findSEL}(\sigma)$ of line 14 returns \emptyset , and job promotion is performed from line 15 to line 17. In this case, no jobs of \mathbf{T}' appear in the current ready queue \mathcal{J}_r . Therefore, when the promoted job can complete before the next arrival, it causes no interference to sub task set \mathbf{T}' , but accrues additional utility to the system.

Combining the above two cases, if promoted jobs can complete before the next arrival, they do not affect the scheduling objectives of sub task set \mathbf{T}' . \square

6 Experimental Results

6.1 Experimental Settings

To evaluate the efficiency of VCUA, we perform simulation experiments to validate our conclusions. We select task sets with 16 tasks in three applications for our study. Their parameters are summarized in Table 1. Within each range, the period P is uniformly distributed. The synthesized task sets simulate the varied mix of short and long periods. The U^{max} s of the TUFs in A_1 , A_2 , and A_3 are uniformly generated within each range. We define a linear increasing VCF = $k \times t + C_o$ for each task. k is uniformed generated within the range $[0, 0.1]$. We change the mean value of C_o , and generate normally-distributed values to adjust the system $load_b$.

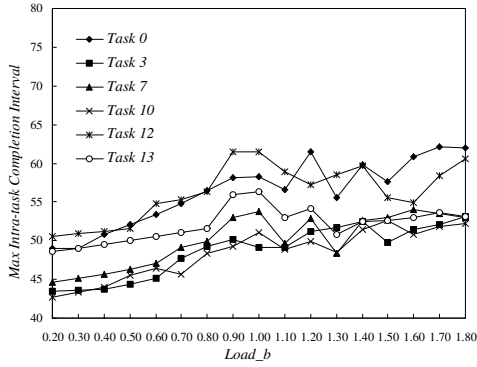
Table 1: Task Settings

Applications	# tasks	Period	U^{max}	$\langle k, C_o \rangle$ (VCF = $k \times t + C_o$)
A_1	4	22 ~ 28	[50, 70]	$\langle 0-0.1, E(C_o) \rangle$
A_2	18	50 ~ 70	[300, 400]	$\langle 0-0.1, E(C_o) \rangle$
A_3	8	2.4 ~ 9.6	[1, 10]	$\langle 0-0.1, E(C_o) \rangle$

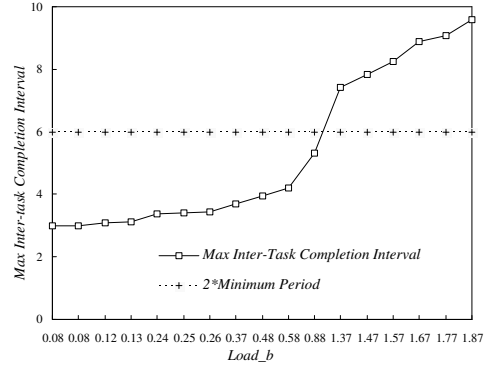
6.2 Performance on Completion Interval

We assign to each task a step TUF, and first consider VCUA's performance on scheduling objective (1). For the 16 tasks, we vary the system $load_b$ from less than 0.1 to larger than 1.8, and evaluate the maximum interval between any two consecutive, successful completions of jobs of each task, and of all tasks. We define the former as maximum intra-task completion interval, and the latter as maximum inter-task completion interval.

Figure 4 shows the maximum intra- and inter-task completion intervals, as $load_b$ varies. In Figure 4(a), we select 6 tasks to study their maximum intra-task completion interval. To validate Theorem 4, we also list the periods of these selected tasks in Table 2. From this figure, we observe that when $load_b$ is less than 1,



(a) Intra-Task



(b) Inter-Task

Figure 4: Maximum Intra- and Inter-Task Completion Interval

the maximum intra-task completion interval of each task is less than the length of twice its period. During overloads, some intra-task completion intervals increase rapidly, since these tasks may have been labeled as *skipped* due to their low PUDs, and they can only complete their jobs sparsely during job promotion.

Table 2: Tasks and Their Periods

Task ID	0	3	7	10	12	13
Period	49	43	44	42	50	48

Figure 4(b) shows the maximum inter-task completion interval of the whole task set. The minimum period of the task set is 3, so we also show the curve of $6 = 2 \times \text{min_period}$ for comparison in this figure. From the figure, we observe that when $load_b$ is less than 1, the maximum inter-task completion interval is less than 6, which is the length of twice minimum period. During overloads, the maximum inter-task completion interval exceeds 6, because some tasks are unselected. Therefore, plots in Figure 4(b) validate Theorem 5.

6.3 Performance on Utility Accrual

We then conduct experiments to study the performance of VCUA on the scheduling objective (2). We still consider step TUFs. Figure 5 shows the accrued utility ratio (or AUR) and termination time meet rate (or XMR) of the algorithms as the system $load_b$ increases. AUR is defined as the ratio of accrued aggregate utility to the maximum possible utility, and XMR is the ratio of the number of jobs meeting their termination times to the total number of jobs releases.

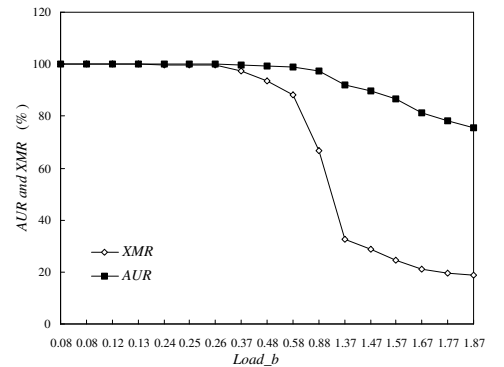
**Figure 5:** AUR and XMR of VCUA with VCF

Figure 5 shows that, when $load_b > 0.4$, VCUA starts to miss termination times and its XMR drops, but its AUR is still near 100%. When $load_b > 0.5$, the AUR also starts to drop. The reason that these two metrics start to decrease to less than 100% even when $load_b \leq 1$ is because that, in our simulation we could not assure that our task settings do not require inserted idles. Thus, we could not validate Corollary 2 from this figure.

On the other hand, Figure 5 also demonstrate the ability of VCUA to accrue utility. During overloads, the XMR drops much faster than AUR, because VCUA favors tasks with higher PUDs in the static selection, and performs job promotion dynamically to utilize excess CPU bandwidth to complete additional jobs, so as to accrue more utility.

6.4 Comparison of VCUA with other Algorithms

Without VCFs, VCUA can be compared with other UA algorithms. In this section, we set the VCFs of VCUA as constant functions, and compare VCUA with three other UA algorithms including DASA [3], LBESA [16], and RUA [20], and a non-UA algorithm EDF without abortion (EDF-NA) [9]. With constant VCFs, $Load_b$ defaults to $Load$. The performance comparisons are shown in terms of AUR and XMR.

We first allocate step TUFs to all tasks, and apply the algorithms on non-preemptive task sets. Figure 6 shows performance comparisons in terms of AUR and XMR of VCUA with the other algorithms. From the figure we can see that during overloads EDF-NA suffers domino effects [16]. On the other hand, although in Figure 6(b) the XMR of VCUA is slightly different from that of the others, it performs as well as the other well-known UA algorithms in terms of AUR in Figure 6(a).

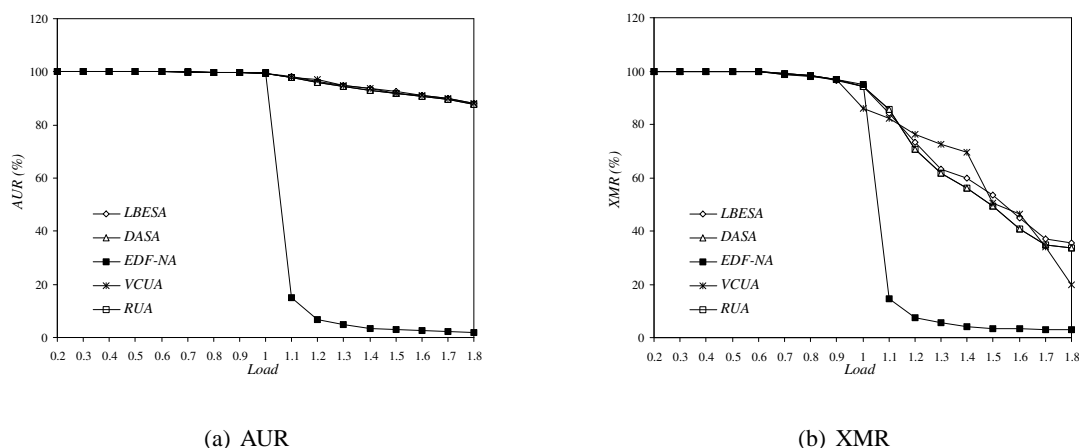


Figure 6: AUR and XMR of VCUA and Other UA Algorithms with Step TUFs

We then repeat the experiments, considering non-step and non-increasing TUFs. Each task is allocated a linear TUF, and its slope is calculated as $-\frac{U^{max}}{P}$, P being the period. Since DASA cannot deal with non-step TUFs, Figure 7 only shows performance comparisons of VCUA with the other three algorithms. With linearly

decreasing TUFs, even if a task is completed before its termination time, it cannot accrue the highest utility U^{max} . Thus, although all mechanisms can attain almost 100% XMRs in Figure 7(b) during under-loads, the corresponding AURs in Figure 7(a) are decreasing, which implies the tasks' response times are getting longer as the system load increases.

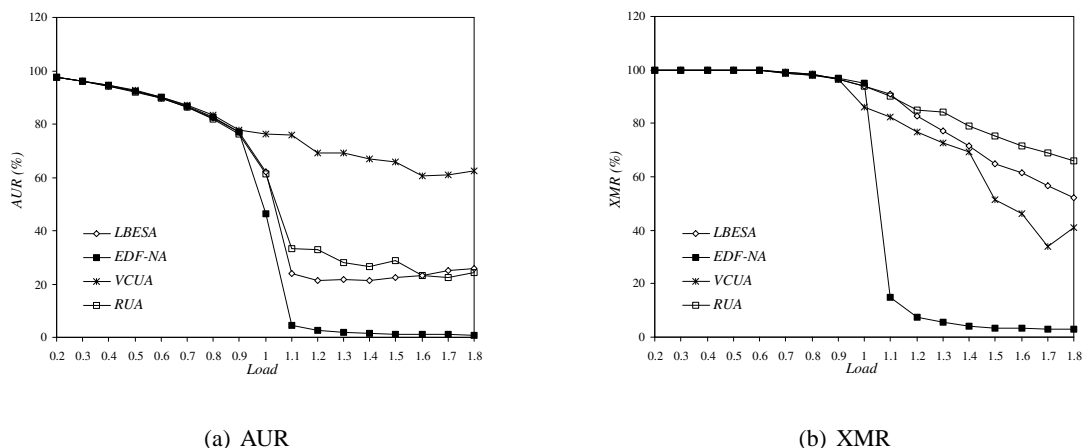


Figure 7: AUR and XMR of VCUA and Other Algorithms with Linearly Decreasing TUFs

We observe similar patterns in the plots of Figure 7 as in those of Figure 6. But with linearly decreasing TUFs, during overloads, VCUA outperforms all the other UA algorithms in terms of AUR, although it yields lower XMR. This is because VCUA can make use of the task-level information in its off-line computing to “select” or “skip” different tasks, favoring tasks with higher PUDs.

7 Conclusions

In this paper, we focus on variable cost scheduling of real-time tasks, and propose VCFs to model tasks' variable execution times that are functions of their starting times. The application activities are subject to time/utility function time constraints and the multi-criteria scheduling objective of assuring that the maximum interval between any two consecutive, successful completion of jobs of a task must not exceed a specified bound, and maximizing the system's total utility. For this problem, we present a utility accrual real-time scheduling algorithm called VCUA. VCUA off-line selects tasks based on their potential utility density, and dynamically promotes jobs to accrue more utility, in polynomial-time. We establish that VCUA achieves optimal timeliness during under-loads, and identify the conditions under which timeliness assurances hold. Our simulation experiments illustrate VCUA's superiority.

Several aspects of the work are directions for further research. Examples include considering resource dependencies among non-periodic and preemptive tasks, considering more complicated shapes of VCFs as

shown in the motivating application examples, and designing non-greedy scheduling algorithms.

References

- [1] K. Chen and P. Muhlethaler. A Scheduling Algorithm for Tasks Described by Time Value Function. *Journal of Real-Time Systems*, 10(3):293–312, May 1996.
- [2] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley. An Adaptive, Distributed Airborne Tracking System. In *Proceedings of The IEEE Workshop on Parallel and Distributed Systems*, volume 1586 of *LNCS*, pages 353–362. Springer-Verlag, April 1999.
- [3] R. K. Clark. *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University, 1990. CMU-CS-90-155, <http://www.real-time.org> (last accessed: January 22, 2005).
- [4] R. K. Clark, E. D. Jensen, and N. F. Rouquette. Software Organization to Facilitate Dynamic Processor Scheduling. In *Proceedings of IEEE Parallel and Distributed Processing Symposium*, April 2004.
- [5] M. Dertouzos. Control Robotics: the Procedural Control of Physical Processes. *Information Processing*, 74, 1974.
- [6] J. K. Dey, J. F. Kurose, and D. Towsley. On-Line Scheduling Policies for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. *IEEE Transactions on Computers*, 45(7):802–813, July 1996.
- [7] GlobalSecurity.org. BMC3I Battle Management, Command, Control, Communications and Intelligence. <http://www.globalsecurity.org/space/systems/bmc3i.htm/> (last accessed: January 22, 2005).
- [8] GlobalSecurity.org. Multi-Platform Radar Technology Insertion Program. <http://www.globalsecurity.org/intell/systems/mp-rtip.htm/> (last accessed: January 22, 2005).
- [9] W. Horn. Some Simple Scheduling Algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.
- [10] K. Jeffay, D. F. Stanat, and C. U. Martel. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. In *Proceedings of The 12th IEEE Real-Time Systems Symposium*, December 1991.
- [11] E. D. Jensen, C. D. Locke, and H. Tokuda. A Time-Driven Scheduling Model for Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 112–122, December 1985.
- [12] G. Koren and D. Shasha. D-Over: An Optimal On-line Scheduling Algorithm for Overloaded Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 290–299, December 1992.
- [13] P. Li. *Utility Accrual Real-Time Scheduling: Models and Algorithms*. PhD thesis, Virginia Tech, 2004. <http://scholar.lib.vt.edu/theses/available/etd-08092004-230138/> (last accessed: January 22, 2005).
- [14] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.
- [15] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung. Imprecise Computations. *Proceedings of the IEEE*, 82(1):83–94, January 1994.
- [16] C. D. Locke. *Best-Effort Decision Making for Real-Time Scheduling*. PhD thesis, Carnegie Mellon University, 1986. CMU-CS-86-134, <http://www.real-time.org> (last accessed: January 22, 2005).
- [17] D. P. Maynard, S. E. Shipman, R. K. Clark, J. D. Northcutt, R. B. Kegley, B. A. Zimmerman, and P. J. Keleher. An Example Real-Time Command, Control, and Battle Management Application for Alpha. Technical report, Department of Computer Science, Carnegie Mellon University, December 1988. Archons Project Technical Report 88121.
- [18] G. W. Stimson. *Introduction to Airborne Radar*. SciTech Publishing, second edition, January 1998.
- [19] J. Wang and B. Ravindran. Time-Utility Function-Driven Switched Ethernet: Packet Scheduling Algorithm, Implementation, and Feasibility Analysis. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):119–133, February 2004.
- [20] H. Wu, B. Ravindran, E. D. Jensen, and U. Balli. Utility Accrual Scheduling under Arbitrary Time/utility Functions and Multi-unit Resource Constraints. In *Proceedings of 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, pages 80–98, August 2004.