Supporting Soft Real-Time DAG-based Systems on Multiprocessors with No Utilization Loss[*]

Cong Liu and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

In work on globally-scheduled real-time multiprocessor systems, analysis is lacking for supporting real-time applications developed using general processing graph models. In this paper, it is shown that bounded deadline tardiness can be ensured for such applications on a multiprocessor with no utilization loss. This result is general: it is applicable to periodic, sporadic, and rate-based directed-acyclic-graph (DAG) models and allows sophisticated notions of precedence to be supported (particularly, notions allowed by the processing graph method*). This paper is the first to show that bounded tardiness can be ensured for globally-scheduled DAG-based applications without utilization loss.*

1 Introduction

In many real-time systems, applications are defined using processing graphs [2,4], where vertices represent sequential code segments and edges represent precedence constraints. For example, signal-processing algorithms are often specified using dataflow graphs [4]. With the advent of multicore technologies, it is inevitable that such applications will be deployed on multiprocessors. Motivated by this, we consider in this paper multiprocessor implementations of systems specified as DAGs.

If all deadlines in such a system are viewed as hard, and tasks execute sporadically (or periodically), then DAG-based systems can be easily supported by assigning a common period to all tasks in a DAG and by adjusting job releases so that successive tasks execute in sequence. Fig. 1 shows an example of scheduling a DAG T_1 on a two-processor system consisting of four sporadic tasks, T_1^1, T_1^2, T_1^3, and T_1^4. (DAG-based systems are formerly defined in Sec. 2. It suffices to know here that the k^{th} job of T_1^1, T_1^2 (or T_1^3), and T_1^4, respectively, must execute in sequence.)

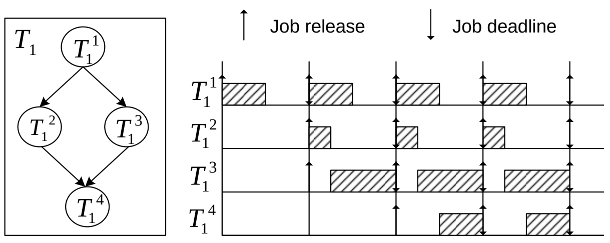


Figure 1. Example DAG.

As seen in this example, the timing guarantees provided by the sporadic model ensure that any DAG executes correctly as long as no deadlines are missed.

However, if all deadlines in a multiprocessor sporadic task system must be viewed as hard, then significant processing capacity must be sacrificed, due to either inherent schedulability-related utilization loss—which is unavoidable under most scheduling schemes—or high runtime overheads—which typically arise in optimal schemes that avoid schedulability-related loss. In systems where less stringent notions of real-time correctness suffice, such loss can be avoided by viewing deadlines as soft. In this paper, such systems are our focus; the notion of soft real-time correctness we consider is that deadline tardiness is bounded.

Bounded deadline tardiness is a notion that has been studied extensively in the context of *global* scheduling algorithms, and such algorithms are our focus as well. Under global scheduling, tasks are scheduled from a single run queue and may migrate across processors (in contrast, under partitioning schemes, tasks are statically bound to processors). It has been shown in recent work that a variety of global scheduling algorithms can ensure bounded deadline tardiness in sporadic task systems with no utilization loss on multiprocessors, including algorithms that are less costly to implement than optimal algorithms [3, 7].[1] Unfortunately, if deadlines can be missed, then DAGs are not as easy to support as ordinary sporadic tasks. For example, if the first

[*]Work supported by AT&T and IBM Corps.; NSF grants CNS 0834270 and CNS 0834132; ARO grant W911NF-09-1-0535; and AFOSR grant FA 9550-09-1-0549.

[1]Technically, bounded tardiness can only be ensured for task systems that do not over-utilize the underlying platform. In all claims in this paper concerning bounded tardiness, a non-over-utilized system is assumed.

job of T_1^1 in Fig. 1 were to miss its deadline, then its execution might overlap that of the first jobs of T_1^2 and T_1^3. This violates the requirement that instances of successive DAG vertices must execute in sequence.

Regarding the state of the art on globally-scheduled real-time multiprocessor systems, analysis is lacking for supporting real-time applications developed using DAG-based formalisms. In this paper, we address this lack of support by presenting scheduling techniques and analysis that can be applied to support DAG-based systems on multiprocessors with no utilization loss, assuming that bounded deadline tardiness is the timing guarantee that must be ensured. Our results can be applied to systems with rather sophisticated precedence constraints. To illustrate this, we consider a particularly expressive DAG-based formalism, the processing graph method (PGM) [6], which was the focus of prior uniprocessor-based research [4]. We show that PGM-specified systems can be scheduled with bounded tardiness on multiprocessors with no utilization loss.

Related work. To our knowledge, DAGs have not been considered before in the context of global real-time scheduling algorithms. However, the issue of scheduling PGM graphs on a uniprocessor was extensively considered by Goddard in his dissertation [4]. (In fact, his work inspired the research in this paper.) Goddard presented techniques for mapping PGM nodes to tasks in the rate-based-execution (RBE) task model [5], as well as conditions for verifying the schedulability of the resulting task set under a rate-based, earliest-deadline-first (EDF) scheduler.

The scheduling of DAGs in distributed systems (which must be scheduled by partitioning approaches) has also been considered. An overview of such work (which we omit here due to space constraints) can be found in [10].

Recently, we showed that bounded tardiness can be ensured for sporadic task systems with pipeline-based precedence constraints under the global earliest-deadline-first (GEDF) scheduling algorithm provided certain conditions hold [8–11]. However, these results only apply to pipelines and require utilization constraints that can be pessimistic.

Contributions. In this paper, we show that sophisticated notions of acyclic precedence constraints can be supported under GEDF on multiprocessors without utilization loss, provided bounded deadline tardiness is acceptable. The types of precedence constraints we consider are those allowed by PGM and studied previously in the uniprocessor case by Goddard [4]. Since any acyclic PGM graph has a natural representation as a DAG-based rate-based (RB) task system, such systems are our major focus. We specifically show that, when any such system is scheduled by GEDF, each task's maximum tardiness is bounded. We show this by transforming any such system into an ordinary sporadic system (with sporadic job releases and without precedence constraints) and by exploiting the fact the latter has bounded

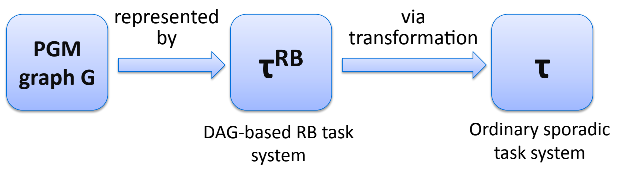


Figure 2. Roadmap.

tardiness under GEDF. Note that, although we focus specifically on GEDF in this paper, our results can be applied to any global scheduling algorithm that can ensure bounded tardiness with no utilization loss for ordinary sporadic task systems. Moreover, due to the fact that our RB task model is a generalization of the periodic and sporadic task models (see Sec. 2), our results are general enough to be applicable to periodic and sporadic DAG systems.

Roadmap. In this paper, we primarily deal with PGM graphs, DAG-based RB task systems, and ordinary sporadic task systems. For clarity, we let G denote a PGM graph, τ denote an ordinary sporadic task system, and τ^{RB} denote a DAG-based RB task system, as shown in Fig. 2.

The rest of this paper is organized as follows. We first present needed definitions in Sec. 2. Then, in Sec. 3, we present our main result. In Sec. 4, we demonstrate the utility of this result via a case study, and in Sec. 5, we conclude.

2 Preliminaries

In this section, we present an overview of the RB task model (where precedence constraints do not arise) and then describe our DAG-based extension of it. We also present an overview of PGM. For a complete description of the PGM, please see [6].

RB task model. The *RB task model* is a general task model in which each task is specified by four parameters: (x, y, d, e). The pair (x, y) represents the maximum execution rate of an RB task: x is the maximum number of invocations of the task in any interval $[j \cdot y, (j+1) \cdot y)$ $(j \geq 0)$ of length y; such an invocation is called a *job* of the task. x and y are assumed to be non-negative integers. Additionally, d is the task's relative deadline, and e is its worst-case execution time. The *utilization* of an RB task is $e \cdot \frac{x}{y}$. It is required that $e \cdot \frac{x}{y} \leq 1$, for otherwise, the system may become overloaded and tasks may have unbounded response times. The widely-studied sporadic task model is a special case of the RB task model. In the sporadic task model, a task is released no sooner than every p time units, where p is the task's period. In the RB task model, the notion of a "rate" is much more general.

Note that the RB task model used in this paper is simi-

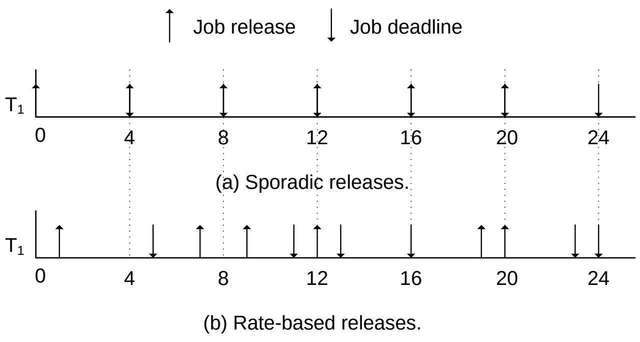


Figure 3. Sporadic and RB releases.

lar to the RBE task model [5], except that **(i)** the RBE task model assumes that x is the number of executions *expected* to be requested in any interval of length y, and **(ii)** the RBE task model specifies a minimum separation between consecutive job deadlines of the same task (our RB model does not require such a minimum separation). Despite these differences, the overall (multiprocessor) scheduling strategy we obtain is similar to that presented previously by Goddard for the uniprocessor case [4]. We elaborate on this point further in Sec. 5.

Example. Fig. 3 shows job release times and deadlines for a task $T_1(1, 4, 4, e)$. In inset (a), jobs are released sporadically, once every four time units in this case. Inset (b) shows a possible job-release pattern that is not sporadic. As seen, the second job is released at time 7 while the third job is released at time 9. The separation time between these jobs is less than seen in the sporadic schedule.

DAG-based RB task model. The DAG-based RB task model extends the RB task model by allowing precedence constraints to exist among tasks/jobs. The exact manner in which such constraints arise is motivated by those seen in acyclic PGM specifications, as we shall see.

A *DAG-based RB task system* is comprised of a set $\tau^{RB} = \{T_1, ..., T_n\}$ of n independent DAG-based RB tasks, which we assume are to be scheduled on $m \geq 2$ identical processors. A z-node *DAG-based RB task*, T_l, consists of z connected RB tasks (or nodes), $T_l^1, ...T_l^z$, which may have different execution rates. (If $z = 1$, then T_l is an ordinary RB task.) Each DAG T_l has a *source node* T_l^1. Between any two connected nodes is an edge. A node can have outgoing or incoming edges. A source node, however, can only have outgoing edges. We assume that any DAG T_l is fully-connected, i.e., any node T_l^h ($h > 1$) is reachable from the source node T_l^1. As before, an RB task T_l^h is released at most x_l^h times in any interval $[j \cdot y_l^h, (j + 1) \cdot y_l^h)$ ($j \geq 0$), with each such invocation called a *job*. The j^{th} job of T_l^h in DAG T_l, denoted $T_{l,j}^h$, is released at time $r^{RB}(T_{l,j}^h)$ and has a deadline at time $d^{RB}(T_{l,j}^h)$. (We will later denote job release times and deadlines using different notation for other task systems.) The relative deadline of T_l^h, denoted d_l^h, is y_l^h/x_l^h. Thus, for any job $T_{l,j}^h$,

$$d^{RB}(T_{l,j}^h) = r^{RB}(T_{l,j}^h) + d_l^h. \tag{1}$$

The *utilization* of each RB task T_l^h in T_l, denoted u_l^h, is $e_l^h \cdot \frac{x_l^h}{y_l^h}$. The *utilization of the task system* τ^{RB} is $U_{sum}(\tau^{RB}) = \sum_{T_i \in \tau^{RB}} \sum_{T_i^j \in T_i} u_i^j$. We define the *depth* of a task to be the number of edges on the longest path between this task and the source task of the corresponding DAG.

Example Fig. 4(a) shows an example DAG-based RB task system with one DAG T_1 containing four tasks. T_1^1 is the source node. T_1^2 and T_1^3 are depth-1 tasks while T_1^4 is a depth-2 task. T_1^1 has two outgoing edges to T_1^2 and T_1^3, and T_1^4 has two incoming edges from T_1^2 and T_1^3. For this system, the total utilization is $u_1^1 + u_1^2 + u_1^3 + u_1^4 = 1/2 + 1/3 + 2/3 + 1/2 = 2$.

Consecutively-released jobs of the same task must execute in sequence (this is the precedence constraint enforced by the RB task model). Also, precedence constraints exist among tasks/jobs within a DAG. Edges represent potential precedence constraints among connected tasks. If there is an edge from task T_l^k to task T_l^h in the DAG T_l, then T_l^k is called a *predecessor task* of T_l^h. We let $pred(T_l^h)$ denote the set of all predecessor tasks of T_l^h.

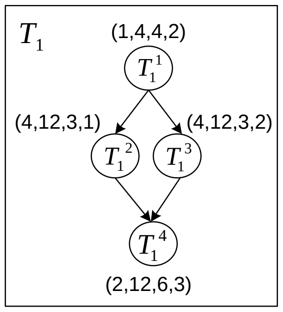
A job $T_{l,j}^h$ may be restricted from beginning execution until certain jobs of tasks in $pred(T_l^h)$ have completed. We denote the set of such *predecessor jobs* as $pred(T_{l,j}^h)$. Defining $pred(T_{l,j}^h)$ precisely requires that job precedence constraints be specified. However, for a general DAG-based RB task system, this is not so straightforward. This is because RB tasks may execute at different rates, and their job release times are not specifically defined (only maximum rates are specified). However, the release time of a job can clearly be no earlier than the release time of any of its predecessor jobs. That is, for any job $T_{l,j}^h$ and one of its predecessor jobs, $T_{l,v}^w$, we have

$$r^{RB}(T_{l,j}^h) \geq r^{RB}(T_{l,v}^w). \tag{2}$$

As we shall see later, under PGM, job precedence constraints can be explicitly determined. We therefore defer further consideration of such constraints until PGM is introduced.[2]

If a job $T_{l,j}^h$ completes at time t, then its *tardiness* is defined as $max(0, t - d^{RB}(T_{l,j}^h))$. A DAG's tardiness is the maximum of the tardiness of any job of any of its tasks.

[2]Simplifications of the DAG-based RB task model exist in which periodic/sporadic execution is assumed and job precedence constraints are straightforward to determine. An example was seen earlier in Fig. 1.



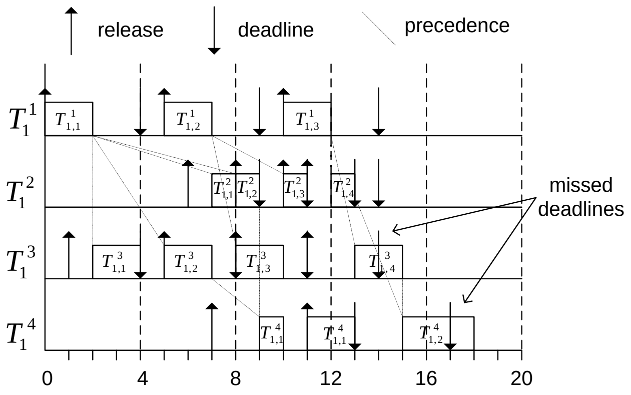


Figure 4. GEDF schedule of the example system.

We require $u_l^h \leq 1$ and $U_{sum}(\tau^{RB}) \leq m$; otherwise, tardiness can grow unboundedly. Note that, when a job of a task misses its deadline, the release time of the next job of that task is not altered. Despite this, it is still required that a job cannot execute in parallel with any of its predecessor jobs or the prior job of the same task.

Under GEDF, released jobs are prioritized by their deadlines. We assume that jobs are ordered by deadline as follows: $T_{i,v}^w \prec T_{a,b}^c$ iff $d^{RB}(T_{i,v}^w) < d^{RB}(T_{a,b}^c)$ or $d^{RB}(T_{i,v}^w) = d^{RB}(T_{a,b}^c) \wedge (i = a) \wedge (w < c)$ or $d^{RB}(T_{i,v}^w) = d^{RB}(T_{a,b}^c) \wedge (i < a)$. $T_{i,v}^w$ has higher priority than $T_{a,b}^c$ iff $T_{i,v}^w \prec T_{a,b}^c$.

Example. Fig. 4(b) shows a two-processor GEDF schedule of the example DAG-based RB task system shown in Fig. 4(a). Regarding the release pattern, note that each task T_1^k releases at most x_1^k jobs within any time interval $[j \cdot y_1^k, (j+1) \cdot y_1^k)$ $(j \geq 0)$. Regarding job precedence constraints, it is assumed in this example that $T_{1,1}^1 = pred(T_{1,1}^2) = pred(T_{1,2}^2) = pred(T_{1,1}^3) = pred(T_{1,2}^3)$, $T_{1,2}^1 = pred(T_{1,3}^2) = pred(T_{1,3}^3)$, $T_{1,3}^1 = pred(T_{1,4}^2) = pred(T_{1,4}^3)$, $pred(T_{1,1}^4) = \{T_{1,2}^2, T_{1,2}^3\}$, and $pred(T_{1,2}^4) = \{T_{1,4}^2, T_{1,4}^3\}$. As seen in the schedule, jobs $T_{1,4}^3$ and $T_{1,2}^4$ miss their deadlines by one time unit. Moreover, $T_{1,2}^4$ can only start execution at time 15 although it is released at time 11. This is because one of its predecessor jobs, $T_{1,4}^3$, completes at time 15. This causes $T_{1,2}^4$ to miss its deadline.

PGM specifications. Like a DAG-based RB task system, an acyclic PGM graph [6] consists of a set of DAGs, each with a distinct source node. Each directed edge in a PGM graph is a typed first-in-first-out (FIFO) queue, and (as before) all nodes in a DAG are assumed to be reachable from the DAG's source node. A producing node transports a certain number of tokens (i.e., some amount of data) to a consuming node, as indicated by the data type of the queue. Tokens are appended to the tail of the queue by the producing node and read from the head by the consuming node. A queue is specified by three attributes: a produce amount, threshold, and consume amount. The produce amount specifies the number of tokens appended to the queue when the producing node completes execution. The threshold amount specifies the minimum number of tokens required to be present in the queue in order for the consuming node to process any received data. The consume amount is the number of tokens dequeued when processing data. The only restriction on queue attributes is that they must be non-negative integral values and the consume amount must be at most the threshold. In the PGM framework, a node is *eligible* for execution when the number of tokens on each of its input queues is over that queue's threshold. Overlapping executions of the same node are disallowed. For any queue connecting nodes G^j and G^k in a PGM graph G, we let $\rho^{k \leftarrow j}$ denote its produce amount, $\varphi^{k \leftarrow j}$ denote its threshold, and $c^{k \leftarrow j}$ denote the consume amount. In the PGM framework, it is often assumed that each source node executes according to a rate-based pattern. Note that, even if all source nodes execute according to a periodic/sporadic pattern, non-source nodes may still execute following a rate-based pattern.

Example. Fig. 5 shows an example PGM graph G_1 containing four nodes. As an example of the notation, each invocation of G_1^1 appends four tokens to the queue shared with G_1^2. G_1^2 may execute, consuming three tokens, when at least seven tokens are in this queue.

3 Supporting PGM-Specified Systems on Multiprocessors

In this section, we present our proposed approach for supporting PGM-specified systems on multiprocessors. We first show that any PGM graph G can be represented by a DAG-based RB task system τ^{RB} by mapping PGM nodes to RB tasks. We then show that τ^{RB} can be transformed to an ordinary sporadic task system τ, for which tardiness bounds can be derived. A summary of the terms defined so far, as well as some additional terms defined later, is presented in Table 1.

m	Number of processors
n	Number of tasks
$U_{sum}(\tau)$	Total utilization of a task system τ
(x_i, y_i)	Execution rate of an RB task T_i, indicating that there are at most x_i job releases within any time interval $[j \cdot y_i, (j+1) \cdot y_i)$ $(j \geq 0)$
$\rho^{k \leftarrow j}$	Produce amount of the queue connecting any two nodes G^j and G^k in a PGM graph G
$\varphi^{k \leftarrow j}$	Threshold of the queue connecting any two nodes G^j and G^k in a PGM graph G
$c^{k \leftarrow j}$	Consume amount of the queue connecting any two nodes G^j and G^k in a PGM graph G
$pred(T_l^h)$	Set of predecessor tasks of task T_l^h
$pred(T_{l,j}^h)$	Set of predecessor jobs of job $T_{l,j}^h$
$F_{max}(pred(T_{l,j}^h))$	Latest completion time among all predecessor jobs of $T_{l,j}^h$
$t_f(T_{l,j}^h)$	Completion time of $T_{l,j}^h$
$\varepsilon(T_{l,j}^h)$	Early-release time of $T_{l,j}^h$

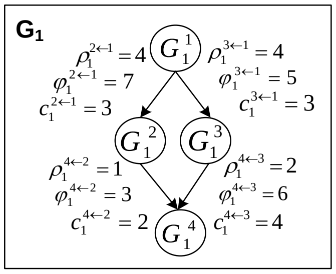
Table 1. Summary of notation.

3.1 Representing PGM Graphs by DAG-based RB Task Systems

In this section, our goal is to implement G by a task system τ^{RB}, where G consists of a set of n acyclic PGM graphs $\{G_1, G_2, ..., G_n\}$. By computing node execution rates (as defined below) based on the producer/consumer relationships that exist in any graph G_l in G, we can implement each node in G_l by an RB task in τ^{RB}.

Definition 1. [4] An *execution rate* is a pair of non-negative integers (x, y). An execution rate specification for any node G_l^i in G_l, (x_l^i, y_l^i), is *valid* if G_l^i releases at most x^i jobs in any time interval $[j \cdot y_l^i, (j+1) \cdot y_l^i)$ $(j \geq 0)$. An execution of a node in G_l is *valid* iff: (1) the task executes only when it is eligible for execution and no two executions of the same node overlap, (2) each input queue has its tokens atomically consumed after each output queue has its tokens atomically produced, and (3) tokens are produced at most once on an output queue during each node execution. An execution of G is *valid* iff all of the nodes in the execution sequence have valid executions and no data loss occurs.

We define τ^{RB} to have the same structure as G, i.e., each



$\mathbf{G_1}$
G_1^1
$\rho_1^{2\leftarrow1}=4$
$\varphi_1^{2\leftarrow1}=7$
$c_1^{2\leftarrow1}=3$
$\rho_1^{3\leftarrow1}=4$
$\varphi_1^{3\leftarrow1}=5$
$c_1^{3\leftarrow1}=3$
G_1^2
G_1^3
$\rho_1^{4\leftarrow2}=1$
$\varphi_1^{4\leftarrow2}=3$
$c_1^{4\leftarrow2}=2$
$\rho_1^{4\leftarrow3}=2$
$\varphi_1^{4\leftarrow3}=6$
$c_1^{4\leftarrow3}=4$
G_1^4

Figure 5. Example PGM graph.

graph G_l in G is implemented by a DAG-based RB task T_l in τ^{RB}. Moreover, each node G_l^i in G_l is implemented by an RB task T_l^i in T_l, and these tasks are connected via edges just like the corresponding nodes in G_l. We assume that the source node of G_l is governed by an RB specification and non-source nodes execute according to the corresponding specifications in G_l (i.e., produce, threshold, and consume attributes of queues in G_l).[3]

As shown in [4], it is possible to compute an execution rate for every task in τ^{RB}. Although the focus of [4] is uniprocessor platforms, this result is independent of the hardware platform. Thus, we first apply the same method to compute an execution rate for every task in τ^{RB}, as stated in the following lemma (proved in [4]).[4]

Lemma 1. *[4] For any task T_i^k in τ^{RB} that has at least one incoming edge, let ν denote the set of predecessor tasks of T_i^k. For any node T_i^u in ν, let $R_i^u = (x_i^u, y_i^u)$ be a valid execution rate. The execution rate $R_i^k = (x_i^k, y_i^k)$ for T_i^k is valid if*

$$y_i^k = lcm\left\{ \frac{c_i^{k\leftarrow v} \cdot y_i^v}{gcd(\rho_i^{k\leftarrow v} \cdot x_i^v, c_i^{k\leftarrow v})} | v \in \nu \right\},$$

$$x_i^k = y_i^k \cdot \frac{\rho_i^{k\leftarrow v}}{c_i^{k\leftarrow v}} \cdot \frac{x_i^v}{y_i^v}, \text{ where } T_i^v \in \nu.\text{[5]}$$

By Lemma 1, we can compute an execution rate (x_i^k, y_i^k) for every task T_i^k in τ^{RB}. Thus, each such task T_i^k can be specified by parameters $(x_i^k, y_i^k, d_i^k, e_i^k)$, where

$$d_i^k = y_i^k / x_i^k. \tag{3}$$

Example. Consider again the example PGM graph shown in Fig. 5. Fig. 6 shows the corresponding DAG-based RB task system. The rate of each task is computed according to Lemma 1, assuming that the source node G_1^1

[3]Given the close connection between τ^{RB} and G, we can henceforth associate tokens, queue attributes, etc., with edges in every DAG-based RB task in τ^{RB}, just like in G.

[4]Note that in Def. 1 and Lemma 1, Goddard actually assumed that each G_l^i releases *exactly* x^i jobs in any time interval $[t + j \cdot y^i, t + (j+1) \cdot y^i)$ $(j \geq 0)$. However, we verified that this definition and lemma generalize to the case where G^i releases *at most* x^i jobs in any such interval.

[5]It is shown in [4] that Def. 1 and Lemma 1 ensure that all tasks in ν with valid execution rates have the same x/y value.

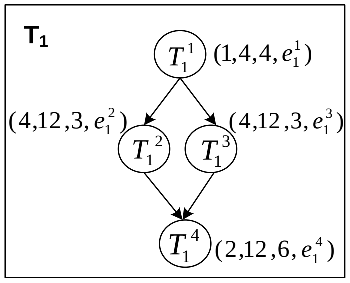


Figure 6. RB counterpart of the PGM graph in Fig. 5.

has an execution rate of $(1,4)$. For instance, $y_1^2 = \dfrac{3 \cdot 4}{gcd(4 \cdot 1, 3)} = 12$ and $x_1^2 = 12 \cdot \dfrac{4 \cdot 1}{3 \cdot 4} = 4$. Also, $y_1^4 = lcm\left\{\dfrac{2 \cdot 12}{gcd(4,2)}, \dfrac{4 \cdot 12}{gcd(8,4)}\right\} = lcm\{12, 12\} = 12$ and $x_1^4 = 12 \cdot \dfrac{1}{2} \cdot \dfrac{4}{12} = 2$. Fig. 7 shows an extended snapshot sequence showing releases of nodes T_1^1 and T_1^2. As seen, T_1^2 releases at most four jobs within any interval $[j \cdot 12, (j+1) \cdot 12)$ $(j \geq 0)$.

In order to completely define τ^{RB}, we need to determine job precedence constraints in τ^{RB}. This is dealt with in the following lemma. In τ^{RB}, any job $T_{i,j}^k$'s predecessor jobs are those that need to complete execution in order for $T_{i,j}^k$ to be eligible to execute.

Lemma 2. *For any $T_i^w \in pred(T_i^k)$, $T_{i,v}^w$ is a predecessor job of $T_{i,j}^k$ iff $v = \left\lceil \dfrac{(j-1) \cdot c_i^{k \leftarrow w} + \varphi_i^{k \leftarrow w}}{\rho_i^{k \leftarrow w}} \right\rceil$.*

Proof. If $T_{i,v}^w$ is a predecessor job of $T_{i,j}^k$, then when $T_{i,v}^w$ completes, the number of tokens in the queue between T_i^w and T_i^k should be at least $\varphi_i^{k \leftarrow w}$ in order for $T_{i,j}^k$ to be able to execute. That is, $v \cdot \rho_i^{k \leftarrow w} - (j-1) \cdot c_i^{k \leftarrow w} \geq \varphi_i^{k \leftarrow w}$ must hold. By rearrangement, we have $v \geq \dfrac{(j-1) \cdot c_i^{k \leftarrow w} + \varphi_i^{k \leftarrow w}}{\rho_i^{k \leftarrow w}}$. Since v must be an integer, we have $v = \left\lceil \dfrac{(j-1) \cdot c_i^{k \leftarrow w} + \varphi_i^{k \leftarrow w}}{\rho_i^{k \leftarrow w}} \right\rceil$. □

Example Consider the example shown in Fig. 7. We can define job precedence constraints according to Lemma 2. For instance, the predecessor job of $T_{1,3}^2$ is $T_{1,v}^1$, where $v = \left\lceil \dfrac{2 \cdot 3 + 7}{4} \right\rceil = 4$, and the predecessor job of $T_{1,4}^2$ is also $T_{1,4}^1$ because $\left\lceil \dfrac{3 \cdot 3 + 7}{4} \right\rceil = 4$.

3.2 Transforming τ^{RB} to τ

We now show that τ^{RB} can be transformed to an ordinary sporadic system τ without utilization loss. The transformation process ensures that all precedence constraints in τ^{RB} are met. Later, in Sec. 3.3, we show that this process ensures that tardiness is bounded for τ^{RB} when GEDF is used.

We transform τ^{RB} to τ by redefining job releases appropriately. First, we must eliminate precedence constraints among tasks within the same DAG. We can do this by redefining job releases so that such constraints are automatically satisfied. By doing so, a DAG can be transformed into a set of independent RB tasks. Second, an RB task may release jobs arbitrarily close together, which is disallowed in the sporadic task model. Therefore, in order to transform τ^{RB} to τ, we also need to re-define job release times to enforce a minimum inter-arrival time.

For any job $T_{l,j}^h$ where $j > 1$ and $h > 1$, its original release time, $r^{RB}(T_{l,j}^h)$, is redefined to be

$$\begin{aligned} r(T_{l,j}^h) \quad &= \quad max\big(r^{RB}(T_{l,j}^h), F_{max}(pred(T_{l,j}^h)), \\ &\qquad r(T_{l,j-1}^h) + d_l^h\big), \end{aligned} \tag{4}$$

where $F_{max}(pred(T_{l,j}^h))$ denotes the latest completion time among all predecessor jobs of $T_{l,j}^h$.

Given that a source task has no predecessors, the release of any job $T_{l,j}^1$ $(j > 1)$ of such a task is redefined to be

$$r(T_{l,j}^1) = max\big(r^{RB}(T_{l,j}^1), r(T_{l,j-1}^1) + d_l^1\big). \tag{5}$$

For the first job $T_{l,1}^h$ $(h > 1)$ of any non-source task, its release time is redefined to be

$$r(T_{l,1}^h) = max\big(r^{RB}(T_{l,1}^h), F_{max}(pred(T_{l,1}^h))\big). \tag{6}$$

Finally, for the first job $T_{l,1}^1$ of any source task, its release time is redefined to be

$$r(T_{l,1}^1) = r^{RB}(T_{l,1}^1). \tag{7}$$

After redefining job releases according to (4)–(7), $T_{l,j}^h$'s redefined deadline, denoted $d(T_{l,j}^h)$, is given by

$$d(T_{l,j}^h) = r(T_{l,j}^h) + d_l^h. \tag{8}$$

Note that these definitions imply that each task's utilization remains unchanged.

Example. Consider the same example as shown in Fig. 6. Fig. 8 shows the redefined job releases of task T_1^2. As seen, according to (4) and (6), $r(T_{1,1}^2) = r^{RB}(T_{1,1}^2)$, $r(T_{1,2}^2) = r(T_{1,1}^2) + d_1^2$, $r(T_{1,3}^2) = r(T_{1,2}^2) + d_1^2$, and $r(T_{1,4}^2) = r^{RB}(T_{1,4}^2)$.

Note that the release time of any job $T_{l,j}^h$ with predecessor jobs is redefined to be at least $F_{max}(pred(T_{l,j}^h))$. Thus, its start time is at least $F_{max}(pred(T_{l,j}^h))$. Hence, the GEDF schedule preserves the precedence constraints enforced by the DAG-based RB model. Note also that, since the release

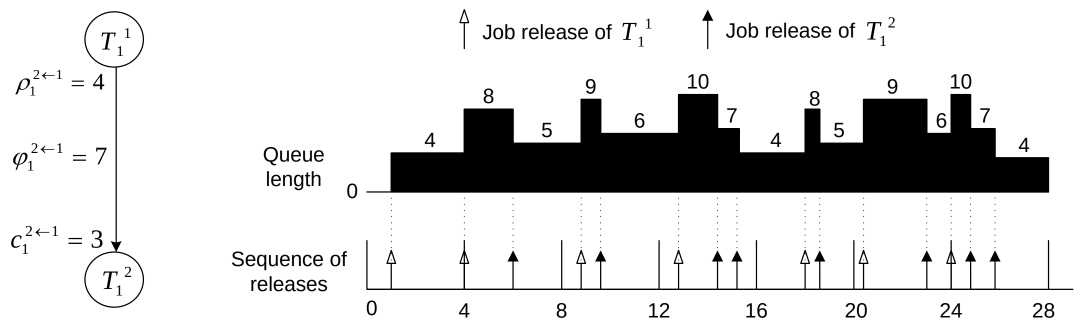
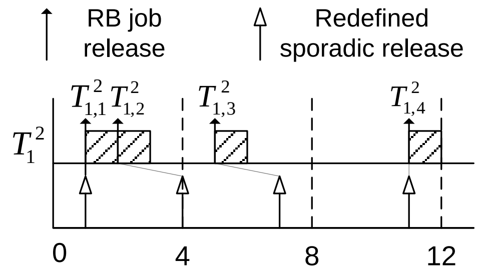


Figure 7. Extended snapshot sequence of releases.



RB job release
Redefined sporadic release
$T_{1,1}^2$ $T_{1,2}^2$ $T_{1,3}^2$ $T_{1,4}^2$
T_1^2
0 4 8 12

Figure 8. Redefining job releases according to (4) – (7).

time of each $T_{l,j}^h$ $(j > 1)$ is redefined to be at least that of $T_{l,j-1}^h$ plus d_l^h, T_l executes as a sporadic task with $p_l^h = d_l^h$. By transforming every task in τ^{RB} to an independent sporadic task according to (4)–(7), we obtain τ. Note that (8) potentially causes job deadlines to move to later points in time.[6] The following lemma establishes an upper bound on the gap between the original deadline and the redefined deadline of any job of any source task. (Actually, it is not necessary to so aggressively shift *releases* to later points in time; this issue is dealt with in Sec. 3.4.)

Lemma 3. *For any job $T_{i,j}^1$, $r(T_{i,j}^1) - r^{RB}(T_{i,j}^1) < 2 \cdot y_i^1$.*

Proof. (All jobs considered in this proof are assumed to be jobs of T_i^1.) It suffices to prove that for any job $T_{i,u}^1$ with $k \cdot y_i^1 \le r^{RB}(T_{i,u}^1) < (k+1) \cdot y_i^1$, where $k \ge -1$, we have $r(T_{i,u}^1) < k \cdot y_i^1 + 2 \cdot y_i^1$. We prove this by induction on k. For conciseness, we make the base case vacuous by starting with $k = -1$ (the base case then holds trivially since no job is released within time interval $[-y_i^1, 0)$).

For the induction step, let us assume that

$$r(T_{i,c}^1) < j \cdot y_i^1 + 2 \cdot y_i^1 \tag{9}$$

holds for any job $T_{i,c}^1$ with $j \cdot y_i^1 \le r^{RB}(T_{i,c}^1) < (j+1) \cdot y_i^1$ $(j \ge 0)$. Then we want to prove that for any job $T_{i,v}^1$ with

$$(j+1) \cdot y_i^1 \le r^{RB}(T_{i,v}^1) < (j+2) \cdot y_i^1, \tag{10}$$

we have $r(T_{i,v}^1) < (j+1) \cdot y_i^1 + 2 \cdot y_i^1$.

Let χ denote the set of jobs with RB release times within $[(j+1) \cdot y_i^1, r^{RB}(T_{i,v}^1)]$. We consider three cases.

Case 1. χ is empty. In this case, $r^{RB}(T_{i,v}^1)$ is the first RB release within $[(j+1) \cdot y_i^1, (j+2) \cdot y_i^1)$. According to (5), either $r(T_{i,v}^1) = r^{RB}(T_{i,v}^1)$ or $r(T_{i,v}^1) = r(T_{i,v-1}^1) + d_i^1$ where $T_{i,v-1}^1$ is the last job of T_i^1 with $j \cdot y_i^1 \le r^{RB}(T_{i,v-1}^1) < (j+1) \cdot y_i^1$. If $r(T_{i,v}^1) = r^{RB}(T_{i,v}^1)$, then $r(T_{i,v}^1) = r^{RB}(T_{i,v}^1) \overset{\text{by (10)}}{<} j \cdot y_i^1 + 2 \cdot y_i^1$. If $r(T_{i,v}^1) = r(T_{i,v-1}^1) + d_i^1$, then by (9), we have $r(T_{i,v-1}^1) < j \cdot y_i^1 + 2 \cdot y_i^1$. Thus, $r(T_{i,v}^1) = r(T_{i,v-1}^1) + d_i^1 < j \cdot y_i^1 + 2 \cdot y_i^1 + d_i^1 \overset{\text{by (3)}}{\le} (j+1) \cdot y_i^1 + 2 \cdot y_i^1$.

Case 2. χ is non-empty and there exists at least one job $T_{i,v'}^1$ in χ such that $r(T_{i,v'}^1) = r^{RB}(T_{i,v'}^1)$. According to (5) and the fact that at most x_i^1 jobs could be released within χ (since $[(j+1) \cdot y_i^1, r^{RB}(T_{i,v}^1)] \in [(j+1) \cdot y_i^1, (j+2) \cdot y_i^1)$), the release time of $T_{i,v}^1$ could be delayed by at most $(x_i^1 - 1) \cdot d_i^1$ time units. By (3), $(x_i^1 - 1) \cdot d_i^1 < y_i^1$. Thus, $r(T_{i,v}^1) < r^{RB}(T_{i,v}^1) + y_i^1 \overset{\text{by (10)}}{<} (j+2) \cdot y_i^1 + y_i^1 = (j+1) \cdot y_i^1 + 2 \cdot y_i^1$.

Case 3. χ is non-empty and there exists no job in χ such that $r(T_{i,v'}^1) = r^{RB}(T_{i,v'}^1)$. In this case, for the first-released job $T_{i,v'}^1$ in χ, we have $r(T_{i,v'}^1) = r(T_{i,v'-1}^1) + d_i^1$. Note that $T_{i,v'-1}^1$ exists, for otherwise, we would have $r(T_{i,v'}^1) = r^{RB}(T_{i,v'}^1)$. Due to the fact that at most $x_i^1 - 1$ jobs could be released within χ, we have $r(T_{i,v}^1) \le r(T_{i,v'-1}^1) + x_i^1 \cdot d_i^1$. Thus, we have $r(T_{i,v}^1) \le r(T_{i,v'-1}^1) + x_i^1 \cdot d_i^1 \overset{\text{by (3) and (9)}}{<} j \cdot y_i^1 + 2 \cdot y_i^1 + y_i^1 = (j+1) \cdot y_i^1 + 2 \cdot y_i^1$. □

3.3 Tardiness Bound for τ^{RB}

Given a DAG-based RB task system, τ^{RB}, by applying the strategy presented above, we obtain a task system τ con-

[6]Note that the method so far yields a non-work-conserving scheduler since the release time of some job may be delayed to a later point of time.

taining only independent sporadic tasks. We can then apply the tardiness bound derived for ordinary sporadic task systems in [3] (or any other such bound), as stated below. Let $t_f(T_{l,j}^h)$ denote the completion time of $T_{l,j}^h$ in τ.

Theorem 1. *[3] In any GEDF schedule for the sporadic task system τ on m processors, if $U_{sum}(\tau) \leq m$, then the tardiness of any job $T_{l,j}^h$, with respect to its redefined deadline $d(T_{l,j}^h)$, is at most Δ, where Δ is an expression depending on system parameters specified in [3] (omitted here due to space constraints), i.e.,*

$$t_f(T_{l,j}^h) - d(T_{l,j}^h) \leq \Delta. \qquad (11)$$

However, Theorem 1 only gives a tardiness bound for any job $T_{l,j}^h$ with respect to its redefined deadline, $d(T_{l,j}^h)$. $T_{l,j}^h$ can have higher tardiness with respect to its original deadline, $d^{RB}(T_{l,j}^h)$. Therefore, we must bound the actual tardiness any job $T_{l,j}^h$ may experience with respect to its original deadline. The following theorem gives such a bound. Let $y_l^{max} = max(y_l^1, y_l^2, ..., y_l^z)$, where z is the number of nodes in T_l. Before stating the theorem, we first prove a lemma that is used in its proof.

Lemma 4. *For any two jobs $T_{l,j}^h$ and $T_{l,k}^h$ of T_l^h in τ^{RB}, where $j < k$, $i \cdot y_l^h \leq r^{RB}(T_{l,j}^h) < (i+1) \cdot y_l^h$ ($i \geq 0$), and $(i+w) \cdot y_l^h \leq r^{RB}(T_{l,k}^h) < (i+w+1) \cdot y_l^h$ ($w \geq 0$), we have $r^{RB}(T_{l,k}^h) - r^{RB}(T_{l,j}^h) > (k-j) \cdot d_l^h - 2 \cdot y_l^h$.*

Proof. (All jobs considered in this proof are assumed to be jobs of T_l^h.) Note that $k-j-1$ denotes the number of jobs other than $T_{l,j}^h$ and $T_{l,k}^h$ released in $[r^{RB}(T_{l,j}^h), r^{RB}(T_{l,k}^h)]$. Depending on the number of such jobs, we have two cases.

Case 1. $k-j-1 \leq 2 \cdot x_l^h - 2$. Given the case condition, $k-j \leq 2 \cdot x_l^h - 1$ holds. Thus, we have $(k-j) \cdot d_l^h - 2 \cdot y_l^h \leq (2 \cdot x_l^h - 1) \cdot d_l^h - 2 \cdot y_l^h \overset{\text{by (3)}}{<} 0$. Since $k > j$, $r^{RB}(T_{l,k}^h) - r^{RB}(T_{l,j}^h) \geq 0 > (k-j) \cdot d_l^h - 2 \cdot y_l^h$.

Case 2. $k-j-1 > 2 \cdot x_l^h - 2$. In this case, more than $2 \cdot (x_l^h - 1)$ jobs other than $T_{l,j}^h$ and $T_{l,k}^h$ are released in $[r^{RB}(T_{l,j}^h), r^{RB}(T_{l,k}^h)]$. By the statement of the lemma, at most x_l^h jobs can be released in $[r^{RB}(T_{l,j}^h), (i+1) \cdot y_l^h)$ or $[(i+w) \cdot y_l^h, r^{RB}(T_{l,k}^h)]$, respectively. Thus,

$$\lambda \geq (k-j-1) - 2 \cdot (x_l^h - 1), \qquad (12)$$

where λ is the number of jobs other than $T_{l,j}^h$ and $T_{l,k}^h$ released in $[(i+1) \cdot y_l^h), (i+w) \cdot y_l^h)$.

Note that the length of the time interval $[(i+1) \cdot y_l^h), (i+w) \cdot y_l^h)$ really depends on the number of jobs released within this interval, due to that fact that at most x_l^h jobs are released within any interval $[(i+1) \cdot y_l^h), (i+2) \cdot y_l^h)$ of length y_l^h. For instance, if k jobs are released within $[(i+1) \cdot y_l^h), (i+w) \cdot y_l^h)$, where $1 \leq k < x_l^h$, then its length is at least y_l^h. If $x_l^h \leq k < 2x_l^h$ jobs are released within this interval, then its length is at least $2 \cdot y_l^h$. In general, by (12), the length of the time interval $[(i+1) \cdot y_l^h), (i+w) \cdot y_l^h)$ is $(i+w) \cdot y_l^h - (i+1) \cdot y_l^h \geq \left\lceil \frac{(k-j-1) - 2 \cdot (x_l^h - 1)}{x_l^h} \right\rceil \cdot y_l^h \geq \frac{(k-j-1) - 2 \cdot (x_l^h - 1)}{x_l^h} \cdot y_l^h = \frac{(k-j) - 2 \cdot x_l^h + 1}{x_l^h} \cdot y_l^h = (k-j) \cdot d_l^h - 2 \cdot y_l^h + d_l^h$.

Given (from the statement of the lemma) that $r^{RB}(T_{l,j}^h) < (i+1) \cdot y_l^h$ and $r^{RB}(T_{l,k}^h) \geq (i+w) \cdot y_l^h$, we have $r^{RB}(T_{l,k}^h) - r^{RB}(T_{l,j}^h) > (i+w) \cdot y_l^h - (i+1) \cdot y_l^h \geq (k-j) \cdot d_l^h - 2 \cdot y_l^h + d_l^h > (k-j) \cdot d_l^h - 2 \cdot y_l^h$. □

Theorem 2. *In any GEDF schedule for τ^{RB} on m processors, if $U_{sum}(\tau^{RB}) \leq m$, then the tardiness of any job $T_{l,j}^h$ of a task T_l^h at depth k, with respect to its original deadline, $d^{RB}(T_{l,j}^h)$, is at most $(k+1) \cdot \Delta + 3(k+1) \cdot y_l^{max}$, i.e.,*

$$t_f(T_{l,j}^h) - d^{RB}(T_{l,j}^h) \leq (k+1) \cdot \Delta + 3(k+1) \cdot y_l^{max}. \qquad (13)$$

Proof. This theorem can be proved by induction on task depth. In the base case, by Theorem 1 and the fact that T_i^1 has no predecessors, its tardiness with respect to its newly-defined deadline, $d(T_{i,j}^1)$, is at most Δ. By Lemma 3, $r(T_{i,j}^1) - r^{RB}(T_{i,j}^1) < 2 \cdot y_i^1$. Thus, with respect to its original deadline, $d^{RB}(T_{i,j}^1)$, $T_{i,j}^1$ has a tardiness bound of $\Delta + 2 \cdot y_i^1 < \Delta + 3 \cdot y_i^{max}$.

For the induction step, let us assume (13) holds for any task T_i^w at depth at most $k-1$, $k \geq 1$. Then, the tardiness of any job $T_{i,v}^w$ of T_i^w is at most $k \cdot \Delta + 3k \cdot y_i^{max}$, i.e.,

$$t_f(T_{i,v}^w) - d^{RB}(T_{i,v}^w) \leq k \cdot \Delta + 3k \cdot y_i^{max}. \qquad (14)$$

We want to prove that for any job $T_{i,j}^h$ of any task T_i^h at depth k, $t_f(T_{i,j}^h) - d^{RB}(T_{i,j}^h) \leq (k+1) \cdot \Delta + 3(k+1) \cdot y_i^{max}$. According to (4) and (6), there are three cases to consider regarding $T_{i,j}^h$'s newly-defined release time $r(T_{l,j}^h)$, as illustrated in Fig. 9.

Case 1. $r(T_{i,j}^h) = r^{RB}(T_{i,j}^h)$. By Theorem 1, we know that $t_f(T_{i,j}^h) - d(T_{i,j}^h) \leq \Delta$. Given that $d(T_{i,j}^h) = d^{RB}(T_{i,j}^h)$, we have $t_f(T_{i,j}^h) - d^{RB}(T_{i,j}^h) \leq \Delta < (k+1) \cdot \Delta + 3(k+1) \cdot y_i^{max}$.

Case 2. $r(T_{i,j}^h) = F_{max}(pred(T_{i,j}^h))$. Let $T_{i,v}^w$ be the predecessor of $T_{l,j}^h$ that has the latest completion time among all predecessors of $T_{i,j}^h$ ($T_{i,v}^w$ exists because the depth of T_l^h is at least one). Thus, we have

$$r(T_{i,j}^h) = F_{max}(pred(T_{i,j}^h)) = t_f(T_{i,v}^w). \qquad (15)$$

Therefore,

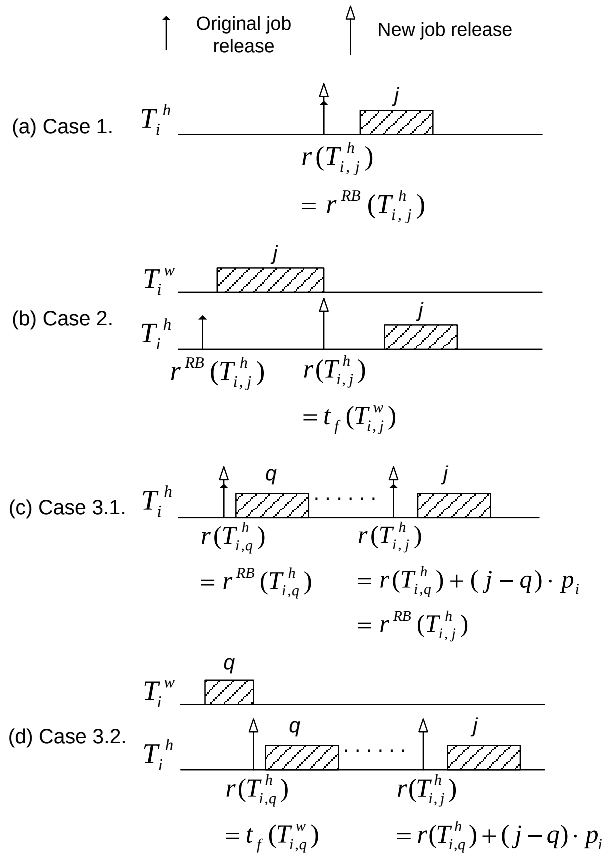


Figure 9. Three cases in Theorem 2.

$$\begin{aligned}
& t_f(T_{i,j}^h) - d^{RB}(T_{i,j}^h) \\
& \{\text{by (1)}\} \\
= \; & t_f(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
= \; & t_f(T_{i,j}^h) - r(T_{i,j}^h) + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
& \{\text{by (8)}\} \\
= \; & t_f(T_{i,j}^h) - d(T_{i,j}^h) + d_i^h + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
& \{\text{by (11)}\} \\
\leq \; & \Delta + d_i^h + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
= \; & \Delta + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) \\
& \{\text{by (2) and (15)}\} \\
\leq \; & \Delta + t_f(T_{i,v}^w) - r^{RB}(T_{i,v}^w) \\
& \{\text{by (1)}\} \\
= \; & \Delta + t_f(T_{i,v}^w) - d^{RB}(T_{i,v}^w) + d_i^w \\
& \{\text{by (14)}\} \\
\leq \; & \Delta + k \cdot \Delta + 3k \cdot y_i^{max} + d_i^w \\
& \{\text{by (3)}\} \\
< \; & (k+1) \cdot \Delta + 3(k+1) \cdot y_i^{max}.
\end{aligned}$$

Case 3. $j > 1 \land r(T_{i,j}^h) = r(T_{i,j-1}^h) + d_i^h$. Let $T_{i,q}^h$ ($q < j$) denote the last job of T_i^h released before $T_{i,j}^h$ such that $r(T_{i,q}^h) = r^{RB}(T_{i,q}^h)$ or $r(T_{i,q}^h) = F_{max}(pred(T_{i,q}^h))$. $T_{i,q}^h$ exists because according to (6) and (7), there exists at least one job, $T_{i,1}^h$, such that $r(T_{i,1}^h) = r^{RB}(T_{i,1}^h)$ or $r(T_{i,1}^h) = F_{max}(pred(T_{i,1}^h))$. Depending on the value of $r(T_{i,q}^h)$, we have two subcases.

Case 3.1. $r(T_{i,q}^h) = r^{RB}(T_{i,q}^h)$. By the definition of $T_{i,q}^h$, the release time of any job $T_{i,k}^h$, where $q < k \leq j$, is redefined to be $r(T_{i,k}^h) = r(T_{i,k-1}^h) + d_i^h$. Thus, we have

$$r(T_{i,j}^h) = r(T_{i,q}^h) + (j-q) \cdot d_i^h. \tag{16}$$

Therefore, we have

$$\begin{aligned}
& t_f(T_{i,j}^h) - d^{RB}(T_{i,j}^h) \\
& \{\text{by (1)}\} \\
= \; & t_f(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
= \; & t_f(T_{i,j}^h) - r(T_{i,j}^h) + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
& \{\text{by (8)}\} \\
= \; & t_f(T_{i,j}^h) - d(T_{i,j}^h) + d_i^h + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
& \{\text{by (11)}\} \\
\leq \; & \Delta + d_i^h + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) - d_i^h \\
= \; & \Delta + r(T_{i,j}^h) - r^{RB}(T_{i,j}^h) \\
& \{\text{by (16) and Lemma 4}\} \\
< \; & \Delta + (r(T_{i,q}^h) + (j-q) \cdot d_i^h) - (r^{RB}(T_{i,q}^h) \\
& +(j-q) \cdot d_i^h - 2 \cdot y_i^h) \\
& \{\text{by the case condition}\} \\
= \; & \Delta + 2 \cdot y_i^h \\
< \; & (k+1) \cdot \Delta + 3(k+1) \cdot y_i^{max}.
\end{aligned}$$

Case 3.2. $r(T_{i,q}^h) = F_{max}(pred(T_{i,q}^h))$. Let $T_{i,v}^w$ denote a predecessor job of $T_{i,q}^h$ with $t_f(T_{i,v}^w) = F_{max}(pred(T_{i,q}^h)) = r(T_{i,q}^h)$. We have

$$\begin{aligned}
& t_f(T_{i,j}^h) - d^{RB}(T_{i,j}^h) \\
& \{\text{similarly to the derivation in Case 3.1}\} \\
< \; & \Delta + (r(T_{i,q}^h) + (j-q) \cdot d_i^h) - (r^{RB}(T_{i,q}^h) \\
& +(j-q) \cdot d_i^h - 2 \cdot y_i^h) \\
= \; & \Delta + r(T_{i,q}^h) - r^{RB}(T_{i,q}^h) + 2 \cdot y_i^h \\
& \{\text{by the case condition and (2)}\} \\
\leq \; & \Delta + t_f(T_{i,v}^w) - r^{RB}(T_{i,v}^w) + 2 \cdot y_i^h \\
& \{\text{by (1)}\} \\
= \; & \Delta + t_f(T_{i,v}^w) - d^{RB}(T_{i,v}^w) + d_i^w + 2 \cdot y_i^h \\
& \{\text{by (14)}\} \\
\leq \; & \Delta + k \cdot \Delta + 3k \cdot y_i^{max} + d_i^w + 2 \cdot y_i^h \\
& \{\text{by (3)}\} \\
\leq \; & (k+1) \cdot \Delta + 3(k+1) \cdot y_i^{max}. \qquad \square
\end{aligned}$$

3.4 Improving Job Response Times by Early-Releasing

By forcing RB releases to be sporadic, we essentially delay job releases. However, excessive release delays are actually unnecessary and actual response times can be improved by

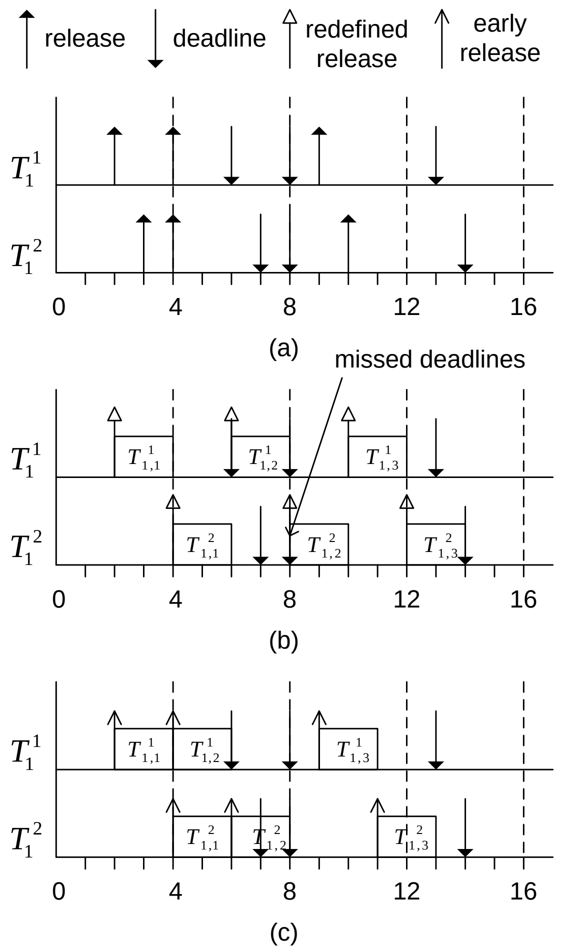


Figure 10. Early-releasing example.

applying a technique called "early-releasing," which allows jobs to execute before their specified release times. The earliest time at which job $T_{l,j}^h$ may execute is defined by its *early-release time* $\varepsilon(T_{l,j}^h)$, where $\varepsilon(T_{l,j}^h) \leq r(T_{l,j}^h)$. For any job $T_{l,j}^h$, its early-releasing time can be defined as

$$\varepsilon(T_{l,j}^h) = \begin{cases} r^{RB}(T_{l,j}^h) & \text{if } h = 1 \\ F_{max}(pred(T_{l,j}^h)) & \text{if } h > 1. \end{cases}$$

An unfinished job $T_{l,j}^h$ is *eligible* for execution at time t if $T_{l,j-1}^h$ has completed by t (if $j > 1$) and $t \geq \varepsilon(T_{l,j}^h)$. As shown in [10], the tardiness bound in Theorem 1 continues to hold if early-releasing is allowed. Intuitively, this is reflective of the fact that schedulability mainly hinges on the proper spacing of consecutive job *deadlines* of a task, instead of its *releases*. This same intuition underlies the development of the uniprocessor RBE model [5].

Example. Consider a DAG-based RB task system scheduled on two processors under GEDF consisting of two tasks: $T_1^1(1,4,4,2)$ and $T_1^2(1,4,4,2)$, where $pred(T_1^2) = T_1^1$ and $pred(T_{2,j}^2) = T_{1,j}^1$ for any $j > 0$. Fig. 10(a) shows the original RB releases before time 12. Fig. 10(b) shows the redefined releases according to (4)–(7), as well as the GEDF schedule. As seen in the schedule, $T_{1,2}^2$ completes at time 10 and misses its original deadline, which is at time 8. Fig. 10(c) shows early releases as defined above and the corresponding GEDF schedule. As seen, most jobs' response times are improved. For instance, $T_{1,2}^2$ now completes at time 8 and meets its original deadline.

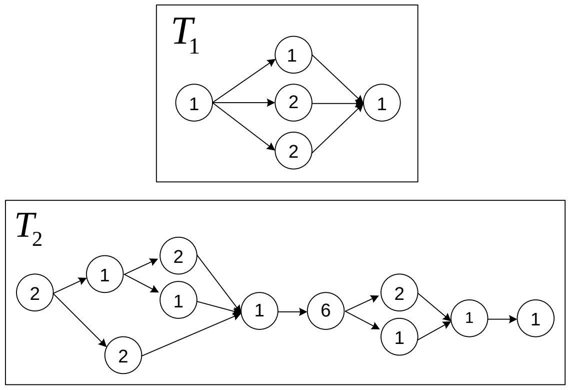


Figure 11. Case study.

	Previous approach	Our approach
Processors needed	**6**	**3**
Max. bound of T$_1$	**0**	**102ms**
Max. bound of T$_2$	**0**	**272 ms**
Max. observed tardiness	**0**	**10 ms**

Table 2. Case study results.

4 Case Study

In this section, we present a case study that demonstrates the utility of our results. Due to space constraints, we only study a simple DAG-based system with two sporadic DAGs, as shown in Fig 11. Each sporadic DAG contains a number of sporadic tasks, with a common period of 10*ms*. Node labels in Fig. 11 give the execution cost (in *ms*) of each task. The total utilization of this system is 3.0. Prior to our work, the only existing approach that could be applied to successfully schedule a DAG-based system under GEDF on multiprocessors is to ensure that every job meets its deadline, so that DAG-based precedence constraints can automatically be satisfied (as seen in Fig. 1).

Table 2 shows a comparison of our approach and this previous approach, where in the latter case, a GEDF schedulability test by Baker [1] was used to ensure that deadlines are not missed. In this table, the number of required processors, theoretical tardiness bounds (computed using Theorems 1 and 2), and the maximum observed tardiness are shown. The latter was determined by simulating the schedule until it repeats. As seen, although our approach requires a higher tardiness bound, we only need three processors to correctly schedule this system under GEDF with bounded tardiness, which is only half of the processors required by the other approach. Moreover, the maximum observed tardiness arising under our approach is low.

5 Conclusion

We have shown that DAG-based systems with sophisticated notions of acyclic precedence constraints can be supported under GEDF on multiprocessors with no utilization loss provided bounded deadline tardiness is acceptable. Our results also imply that any global scheduling algorithm that can ensure bounded tardiness with no utilization loss for ordinary sporadic task systems can ensure the same for any DAG-based task system. Our results are general enough to be applicable to periodic/sporadic DAG-based systems as well.

Our overall scheduling strategy is similar to that presented previously by Goddard for the uniprocessor case [4]. However, several differences do exist. First, under the RBE task model used in [4], a rate is specified by parameters x and y, where x is the number of executions expected to be requested in any interval of length y; contrastingly, in our RB task model, x is the maximum number of executions in an interval of length y. However, Goddard assumes that the source node of a PGM graph executes according to a deterministic rate-based pattern (i.e., exactly x executions in any interval y), so this difference becomes immaterial. Second, the RBE task model specifies a minimum separation between consecutive job deadlines of the same task. Although our RB task model does not require such a minimum separation, we ultimately redefine releases and deadlines to enforce a minimum separation (Sec. 3.2), and early-release jobs to obtain a work-conserving scheduler (Sec. 3.4). Third, in the uniprocessor case, DAG-based precedence constraints can be more easily eliminated than in the multiprocessor case. In [4], DAG-based precedence constraints are met by keeping the job ready queue in EDF order and breaking deadline ties on a FIFO basis. We instead redefine job releases to eliminate all DAG-based precedence constraints (see (4)–(7)). Finally, after mapping PGM graphs to rate-based tasks, Goddard derives a schedulability condition for the resulting system. We instead further transform rate-based tasks into sporadic tasks and derive a schedulability condition for the sporadic task system. Despite these differences, our approach can be seen as a multiprocessor counterpart of that in [4] for scheduling acyclic PGM graphs.

In future work, we would like to investigate support for cyclic graph-based systems on multiprocessors. Also, as the case study in Sec. 4 suggests, tardiness can be lessened by using more processors. Intermediate choices between the two extremes of having no tardiness and using the fewest processors warrant further study. Finally, it would be interesting to generalize our DAG-based task model to allow multiple sources per DAG.

Acknowledgement: We are grateful to Steve Goddard for his feedback on earlier drafts of this paper.

References

[1] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. of the 24th Real-Time Sys. Symp.*, pp. 120-129, 2003.

[2] S. Chakraborty, T. Erlebach, and L. Thiele. On the complexity of scheduling conditional real-time code. In *Lecture in Computer Science*, Vol. 2125, pp. 38-49, 2001.

[3] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proc. of the 26th IEEE Int'l Real-Time Sys. Symp.*, pp. 330-341, 2005.

[4] S. Goddard. *On the Management of Latency in the synthesis of real-time signal processing systems from processing graphs*. PhD thesis, The University of North Carolina at Chapel Hill, 1998.

[5] K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proc. of the 20th IEEE Int'l Real-Time Sys. Symp.*, pp. 304-314, 1999.

[6] Naval Research Laboratory. Processing Graph Method Specification, prepared by the Naval Research Laboratory for use by the Navy Standard Signal Processing Program Office (PMS-412). 1987.

[7] H. Leontyev and J. Anderson. Tardiness bounds for FIFO scheduling on multiprocessors. In *Proc. of the 19th Euromicro Conf. on Real-Time Sys.*, pp. 71-80, 2007.

[8] C. Liu and J. Anderson. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems. In *Proc. of the 16th IEEE Real-Time and Embedded Tech. and Apps. Symp.*, pp. 23-32, 2010.

[9] C. Liu and J. Anderson. Supporting pipelines in soft real-time multiprocessor systems. In *Proc. of the 21st Euromicro Conf. on Real-Time Sys.*, pp. 269-278, 2009.

[10] C. Liu and J. Anderson. Supporting sporadic pipelined tasks with early-releasing in soft real-time multiprocessor systems. In *Proc. of the 15th IEEE Int'l Conf. on Embedded and Real-Time Computing Sys. and Apps.*, pp. 284-293, 2009.

[11] C. Liu and J. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proc. of the 30th Real-Time Sys. Symp.*, pp. 425-436, 2009.