A generalized parallel task model for recurrent real-time processes

Sanjoy Baruah*, Vincenzo Bonifaci†, Alberto Marchetti-Spaccamela‡, Leen Stougie§, Andreas Wiese‡
*University of North Carolina, Chapel Hill, USA. Email: baruah@cs.unc.edu
†IASI-CNR, Rome, Italy. Email: vincenzo.bonifaci@iasi.cnr.it
‡Sapienza University of Rome, Italy. Email: alberto@dis.uniroma1.it, wiese@dis.uniroma1.it
§Vrije Universiteit Amsterdam and CWI, The Netherlands. Email: l.stougie@vu.nl

***Abstract*—A model is considered for representing recurrent precedence-constrained tasks that are to execute on multiprocessor platforms. A recurrent task is specified as a directed acyclic graph (DAG), a period, and a relative deadline. Each vertex of the DAG represents a sequential job, while the edges of the DAG represent precedence constraints between these jobs. All the jobs of the DAG are released simultaneously and need to complete execution within the specified relative deadline of their release. The task may release jobs in this manner an unbounded number of times, with successive releases occurring at least the specified period apart. The scheduling problem is to determine whether such a recurrent task can be scheduled to always meet all deadlines upon a specified number of processors that are dedicated for the use of this task.**

This problem is shown to be computationally intractable, but amenable to efficient approximate solutions. EDF is shown to be a good approximate scheduling algorithm. Polynomial and pseudo-polynomial schedulability tests, of differing effectiveness, are presented for determining whether a given task can be scheduled by EDF to always meet all deadlines on a specified number of processors.

I. Introduction

Many real-time systems can be modeled as being composed of a finite number of independent recurrent processes or tasks, each of which generates a potentially infinite sequence of jobs. Different formal models have been proposed for representing such recurrent tasks; these models differ from one another in the restrictions they place on the jobs that may be generated by a single task.

Since its origins in the late 1960's and early 1970's [10], [9], [8], [2], real-time scheduling theory has primarily been concerned with determining how multiple recurrent tasks can be scheduled on a shared uni- or multi-processor platform. As processor manufacturers seek to provide large improvements in performance without corresponding increases in power and energy requirements, however, scaling trends in processor design have tended to move away from increasing clock frequencies to increasing the number of cores per processor. This is a continuing trend, with no immediate end in sight. The exact form that the resulting massively parallel multicore CPU's will take has not yet been determined (Will all the cores be identical, or will different cores be specialized to form different functions? Will some be dedicated to certain functionalities, with the rest being general-purpose processors? What kinds of on-chip connectivity between the cores will there be?); however, it seems likely that individual tasks will be allowed to execute exclusively upon dedicated cores or groups of cores. Such an execution environment allows for the possibility of having more expressive task models than the relatively simple recurrent task models considered thus far in the real-time scheduling literature (see, e.g., [12], [13] for an excellent survey on models for representing recurrent real-time tasks). Such new models may allow for partial parallelism within a task, as well as for precedence dependencies between different parts of the task. Unfortunately, such models are not particularly well understood; as Saifullah et al. [11] recently observed, "the growing importance of parallel task models for real-time applications poses new challenges to real-time scheduling theory that has mostly focused on sequential task models."

In this paper, we study a parallel task model that we call the *sporadic DAG model*. Each recurrent task in this model is modeled as a directed acyclic graph (DAG) $G = (V, E)$ which executes upon a platform consisting of m identical processors that are dedicated to the exclusive use of this particular task. Each vertex $v \in V$ of the DAG corresponds to a sequential job, and is characterized by a worst-case execution time (WCET) p_v. Each (directed) edge of the DAG represents a precedence constraint: if (v, w) is a (directed) edge in the DAG then the job corresponding to vertex v must complete execution before the job corresponding to vertex w may begin execution. Groups of jobs that are not constrained (directly or indirectly) by precedence constraints in such a manner may execute in parallel if there are processors available for them to do so. The task is further characterized by a (relative) deadline parameter D and a period T. The interpretation of these parameters is as follows. We say the task releases a *dag-job* at time-instant t when it becomes available for execution. When this happens, we assume that all $|V|$ of the jobs become available for execution simultaneously, subject to the precedence constraints. During any given run the task may release an unbounded sequence of dag-jobs; all $|V|$ jobs that are released at some time-instant t must complete execution by time-instant $t + D$. A minimum interval of duration T must elapse between successive releases of dag-jobs. In this paper we study the setting of one sporadic DAG task.

This research. It has long been known [15] that the preemp-

tive scheduling of a given collection of precedence-constrained jobs (i.e., DAGs) on a multiprocessor platform is NP-hard in the strong sense; this intractability result is easily seen to hold for the sporadic DAG model as well. However, very efficient approximation algorithms (discussed in Section IV-A) that have bounded deviation from optimal behavior are known for scheduling non-recurrent DAGs. While these approximation algorithms are easily extended to deal with the scheduling of sporadic DAG tasks for which the deadline parameter D is no larger than the period parameter T (see Section IV-A for details), they do not, in general, extend to the case when $D > T$. Much of the research described in this paper is therefore concerned with dealing with this case. We show (in Section V) that the "synchronous arrival sequence", in which successive dag-jobs are released exactly the period T time-units apart, does not necessarily correspond to the worst-case behavior of a sporadic DAG task; hence, we cannot determine schedulability properties by simply studying this one behavior of the task. We then consider the earliest deadline first (EDF) scheduling [8], [2] of sporadic DAG tasks on identical multiprocessors. We show that EDF has a *speedup bound* (this metric is formally defined in Section II below) no larger than 2 for scheduling sporadic DAG tasks. We derive two different sufficient schedulability tests for determining whether EDF can schedule a given sporadic DAG task upon a specified identical multiprocessor to always meet all deadlines. These tests have different run-time complexity—one has polynomial run-time while the other has run-time pseudo-polynomial in the representation of the task—and effectiveness (as quantified, again, by the speedup bound metric).

The remainder of this paper is organized as follows. In Section II, we formally define the notation and terminology used in describing our task model. We also formalize the concepts of feasibility, schedulability, and schedulability tests, and the speedup bound metric. We briefly survey some related work in Section III. In Section IV we show that the problems we seek to solve are highly intractable; hence, we are unlikely to be able to obtain efficient algorithms for solving them exactly. (This fact provides justification for the fact that most of our algorithms provide approximate solutions, rather than exact ones.) In Section V we show that the synchronous arrival sequence of a sporadic DAG task need not represent its worst-case behavior. We present, and evaluate, a polynomial-time EDF schedulability test in Section VI, and a pseudo-polynomial one in Section VII. Also in Section VII, we show that the speedup bound of EDF is no larger than 2 for scheduling sporadic DAG tasks.

II. Model and definitions

In the ***sporadic DAG*** model, a task is specified as a 3-tuple (G, D, T), where G is a directed acyclic graph, and D and T are positive integers.

- The DAG G is specified as $G = (V, E)$, where V is a set of vertices and E a set of directed edges between these vertices (it is required that these edges do not form any cycle). Each $v \in V$ denotes a sequential operation (a "job"). Each job $v \in V$ is characterized by a processing time (also known as *worst-case execution time* or WCET) $p_v \in \mathbb{N}$. The edges represent dependencies between the jobs: if $(v_1, v_2) \in E$ then job v_1 must complete execution before job v_2 can begin execution. (We say a job becomes *eligible* to execute once all its predecessor jobs have completed execution.)
- A *period* $T \in \mathbb{N}$. A *release* or arrival of a *dag-job* of the task at time-instant t means that all $|V|$ jobs $v \in V$ are released at time-instant t. The period denotes the minimum amount of time that must elapse between the release of successive dag-jobs: if a dag-job is released at t, then the next dag-job may not be released prior to time-instant $t + T$.
- A *deadline* $D \in \mathbb{N}$. If a dag-job is released at time-instant t then all $|V|$ jobs that were released at t must complete execution by time-instant $t + D$.

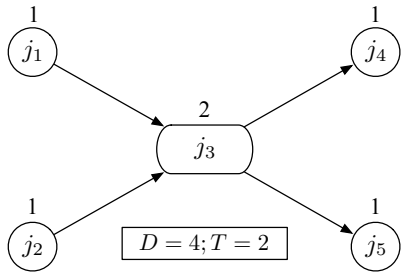


Fig. 1. An example sporadic DAG task. The number above each vertex denotes the WCET of the corresponding job.

Throughout this paper we assume that the input consists of one sporadic DAG task. If $D > T$, the task may release a dag-job prior to the completion of all jobs of the previously-released dag-jobs. We do *not* require that all jobs of a dag-job complete execution before jobs of the next dag-job can start executing.

Some additional notation and terminology:

- A *chain* in the sporadic DAG task (G, D, T) is a sequence of nodes $v_1, v_2, \ldots, , v_k$ such that (v_i, v_{i+1}) is an edge in G, $1 \leq i < k$. The length of this chain is defined to be the sum of the WCETs of all its nodes: $\sum_{i=1}^{k} p_{v_k}$.
- We denote by $\mathrm{len}(G)$ the length of the longest chain in G. Note that $\mathrm{len}(G)$ can be computed in time linear in the number of vertices and the number of edges in G, by first obtaining a topological sorting of the vertices of the graph and then running a straightforward dynamic program.
- We define $\mathrm{vol}(G) = \sum_{v \in V} p_v$. That is, $\mathrm{vol}(G)$ is the total WCET of each dag-job. It is evident that $\mathrm{vol}(G)$ can be computed in time linear in the number of vertices in G.

Example 1. An example sporadic DAG task is depicted in Figure 1. The DAG G for this task consists of five vertices labeled $\{j_1, \ldots, j_5\}$, and four directed edges denoting prece-

dence constraints. Observe that $D > T$ for this task. The longest chain is of length $\mathrm{len}(G) = 4$; the sequence of nodes j_1, j_3, j_5 is an example of such a longest chain. By summing all the WCET's, we can see that $\mathrm{vol}(G) = 6$.

Feasibility and schedulability. Since the period parameter T of the sporadic DAG task (G, D, T) specifies the minimum, rather than exact, duration that must elapse between the release of successive dag-jobs, it is evident that a sporadic DAG may generate infinitely many different collections of dag-jobs. A sporadic DAG is said to be *feasible* on m speed-s processors if a schedule exists on m speed-s processors for every collection of dag-jobs that may be generated by the sporadic DAG. For a given scheduling algorithm A, the sporadic DAG is said to be A-*schedulable* on m speed-s processors if algorithm A meets all deadlines when scheduling every collection of dag-jobs that may be generated by the sporadic DAG on m speed-s processors.

For scheduling algorithm A an A-*schedulability condition* (or *test*) determines, for a given sporadic DAG task (G, D, T) and a specified number m of speed-s processors, whether (G, D, T) is A-schedulable on m speed-s processors. An *exact* A-schedulability test correctly identifies all A-schedulable tasks; a *sufficient* test identifies some but not necessarily all A-schedulable tasks.

Speedup bounds. We will see (Section IV) that the sporadic DAG scheduling problem is highly intractable: NP-hard in the strong sense. It is therefore highly unlikely that we will be able to design efficient algorithms for solving the problem exactly, and our objective is therefore to come up with efficient algorithms that solve the problem *approximately*. A metric we will use for quantifying the quality of an approximation algorithm is its speedup bound:

Definition 1 (**Speedup bound**). An algorithm A for scheduling sporadic DAGs is said to have a *speedup bound* b ($b \geq 1$) if any task that is feasible on m speed-1 processors is A-schedulable on m speed-b processors.

An A-schedulability test is said to have a *speedup bound* b if any task that is feasible on m speed-1 processors is determined by the test to be A-schedulable on m speed-b processors. (Note that a sufficient, but not exact, A-schedulability test may have a larger speedup bound than the speedup bound of algorithm A.)

III. Related Research

There are several models that have been proposed to study the parallel execution of threads in real-time task systems. In [6] the authors model the fork-join parallelism of a task as a sequence of sequential and parallel segments representing the sequential execution of the main thread and of parallel threads respectively. Namely, when the main thread of the task forks into many parallel threads during the execution, then a sequential segment forks in many parallel segments; when the parallel threads finish their execution, the corresponding segments join the sequential segment representing the main thread. In [6] a stretching algorithm has been proposed to transform parallel tasks into sequential tasks when possible.

The model of [11] is the closest to the one studied in this paper and generalizes the fork join model. The authors introduced a DAG model of tasks, where each job is made up of nodes that represent work, and edges that represent dependences between nodes. Therefore, a node can execute only after all of its predecessors have been executed. They propose a decomposition algorithm to assign local deadlines for the different parallel segments and to transform them into sporadic sequential tasks. In the case of implicit deadline tasks they also provide a speed-up bound for this decomposition algorithm of 4 when the tasks are scheduled using global EDF and of 5 for partitioned deadline monotonic scheduling, respectively.

In [12] the authors introduce a model that uses directed graphs to represent the release structure of jobs in terms of order and timing. Namely a task is represented by a directed graph where each vertex represents one type of job that can be released by the task and edges represent the order in which jobs generated by the task are released. The main result is to show that the feasibility problem can be decided in pseudopolynomial time. In [13] the authors study the computational complexity of the extension to the model which allows to express global inter-release separation constraints between non-adjacent job releases. Both [12], [13] consider uniprocessor platforms only.

We finally observe that the new demand function that we propose in Section VII can be reinterpreted as an extension to parallel workloads of the demand function introduced in [1], which was designed for the setting of sequential tasks.

IV. Computational complexity

The uniprocessor version of the sporadic DAG task feasibility problem – given a sporadic DAG task (G, D, T), determine whether it can be scheduled on a single preemptive speed-s processor – is quite tractable: it is straightforward to show that a necessary and sufficient condition is that

$$\frac{\mathrm{vol}(G)}{\min(D, T)} \leq s.$$

If this condition evaluates to true, then an earliest deadline first (EDF) strategy – at each instant, schedule some eligible job belonging to the dag-job with the earliest deadline – will schedule the system to always meet all deadlines. It hence follows that the above condition is also an exact EDF-schedulability test on preemptive uniprocessors.

However, the multiprocessor scheduling of sporadic DAG tasks is highly intractable. When $D \leq T$, determining feasibility is easily seen to be equivalent to the makespan minimization problem for preemptive scheduling of a set of precedence constrained jobs on identical processors, or $P|prec, pmtn|C_{\max}$ in scheduling notation [5]. Therefore, the problem is NP-hard in the strong sense and remains so even when the scheduler is given a speedup $4/3 - \epsilon$, for any $\epsilon > 0$, by a result of Lenstra and Rinnooy Kan [7].

Fact 1. The sporadic DAG feasibility problem is NP-hard in the strong sense. Under the assumption that $\mathsf{P} \neq \mathsf{NP}$, no polynomial-time algorithm for scheduling sporadic DAGs can have a speedup bound smaller than $(4/3 - \epsilon)$ for any $\epsilon > 0$.

In fact, assuming a reasonable complexity-theoretic conjecture that is somewhat stronger than $\mathsf{P} \neq \mathsf{NP}$, a recent result of Svensson [14] implies that even a speedup $(2-\epsilon)$ polynomial time test is ruled out, for any $\epsilon > 0$.

We remark that, when the number of processors is not part of the input (i.e., it is a fixed constant), the complexity of the feasibility problem is related to that of a long standing open problem, known in scheduling notation as $Pm|prec, p_j = 1|C_{\max}$, which appears as open problem "OPEN8" from the original list of Garey and Johnson [3] and is still open.

A. The case $D \leq T$

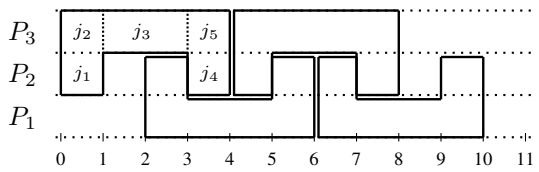
If the relative deadline D of a sporadic DAG is no larger than its period T, then all the jobs of one dag-job must have completed execution before the next dag-job is released. The problem of scheduling the sporadic DAG therefore reduces to determining whether each individual dag-job can be scheduled on the available processors to complete within D time units of the dag-job's release, and is thus equivalent to the makespan minimization problem for preemptive scheduling of a set of precedence constrained jobs on identical processors (the $P|prec, pmtn|C_{\max}$ problem in scheduling notation [5]). It is known that Graham's *list scheduling* algorithm (LS) [4], which is easily implemented to have polynomial run-time, has a speedup bound of 2 for this problem; in conjunction with the recent result of Svensson [14] cited above, this is likely to be the best we can do in polynomial time. For the remainder of this paper, we therefore restrict our attention to sporadic DAGs that have relative deadline greater than period ($D > T$).

V. Identifying the worst-case behavior

Since a sporadic DAG may legally generate infinitely many different collections of dag-jobs, it would be helpful to be able to identify a single collection as representing the "worst-case" collection, in the sense that if this worst-case collection is feasible (respectively, A-schedulable by some algorithm A), then all legal collections are also feasible (resp., A-schedulable). One reasonable candidate for the status of such worst-case behavior is the *synchronous arrival sequence*, in which successive dag-jobs arrive as soon as they are able to do so – i.e., exactly T time units apart. (Intuitively, the synchronous arrival sequence is a reasonable candidate since it maximizes the amount of execution that needs to be completed over a given interval.) However, the synchronous arrival sequence does not consider the parallelism between different jobs; it turns out that as a consequence the synchronous arrival sequence need not in fact correspond to the worst-case behavior of a sporadic DAG:

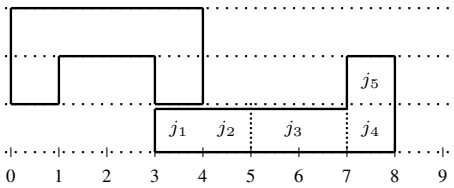
Lemma 1. A sporadic DAG task might be infeasible on m processors even when its synchronous arrival sequence is feasible on m processors.

Proof: Consider again the sporadic DAG task of Figure 1. Suppose we wish to schedule this task on 3 processors. We present below a schedule in Gantt-chart like notation for the synchronous arrival sequence.



As the schedule demonstrates, the synchronous arrival sequence is feasible. Jobs j_1 and j_2 first execute in parallel for one time unit. This is followed by job j_3 executing sequentially for two time units, after which jobs j_4 and j_5 execute in parallel for one time unit. All jobs complete execution within four time units of the release of the DAG-job to which they belong.

However, if one dag-job is released at time 0 and the second dag-job is released at time 3 (instead of 2), then one of the two dag-jobs cannot complete on time:



The problem, as illustrated in the schedule above, is that if the first dag-job completes by its deadline then the second dag-job cannot exploit the parallelism of j_1 and j_2 by executing them in parallel. This requires that these jobs of the second dag-job execute sequentially one after the other, thereby causing the dag-job to miss its deadline (which is at time-instant 7). □

VI. A polynomial-time EDF-schedulability test

In this section, we present a simple sufficient test for determining whether a given sporadic-DAG task (G, D, T) is EDF-schedulable on m unit-speed processors, and show that this test can be implemented efficiently to have polynomial run-time. In Section VI-A we adapt this test to answer the related question: given sporadic-DAG task (G, D, T), how many processors do we need to assign to it in order to ensure that it is EDF-schedulable on the assigned processors? In Section VI-B we show that this test has a speedup bound of $(3 - 1/m)$; in Section VI-C we modify our test so that the modified test has an improved speedup bound of 2.5.

We consider a canonical implementation of EDF: at each point in time t, consider the set of eligible jobs, i.e., the jobs that have not yet completed execution but for which all predecessors in G have already completed execution. If the number of eligible jobs is $\leq m$, then we schedule them all. Otherwise we schedule the m eligible jobs with earliest absolute deadlines, ties broken arbitrarily.

Since we are assuming that $D > T$, we observe that

$$2D > \lceil D/T \rceil T \tag{1}$$

Theorem 1 below provides a sufficient schedulability test for EDF.

Theorem 1. Any sporadic DAG (G, D, T) with $D > T$ satisfying

$$(m-1)\frac{\text{len}(G)}{D} + 2\frac{\text{vol}(G)}{T} \leq m \tag{2}$$

is EDF-schedulable on m unit-speed processors.

Proof: Suppose that EDF fails to meet all deadlines while scheduling some sequence of dag-jobs released by (G, D, T). Consider the first job j which misses its deadline d_j. Without loss of generality we may assume that there are no jobs with a deadline after d_j. Consider the time interval $I = [r_j, d_j)$ — during this interval, EDF is only executing jobs with deadlines within I. Denote by X the total amount of time during I where all m processors are busy. Let $Y := (d_j - r_j) - X$, i.e., Y denotes the total amount of time during I where at least one processor is idle. Consider any longest chain in G and let $V' \subseteq V$ denote all nodes on the chain. Observe that $\text{len}(G) = \sum_{j \in V'} p_j$.

Let x_1 denote the ratio $\text{len}(G)/D$, and x_2 the ratio $\text{vol}(G)/T$.

The crucial observation is that at each instant during I when not all processors are busy, EDF must be executing some job released at time r_j corresponding to a node in V'. Hence, $Y \leq \text{len}(G) \leq x_1 D$. The total work performed by EDF during I is at least

$$\begin{aligned}
& mX + Y \\
&= m(D-Y) + Y \\
&= mD - (m-1)Y \\
&\geq mD - (m-1)\text{len}(G) \text{ (Since } Y \leq \text{len}(G)) \\
&= mD - (m-1)x_1 D \text{ (By definition of } x_1) \\
&= (m - (m-1)x_1)D \\
&> (m - (m-1)x_1)\frac{\lceil D/T \rceil}{2} T \text{ (By (1) above)} \\
&= (m - (m-1)x_1)\frac{\lceil D/T \rceil}{2}\frac{\text{vol}(G)}{x_2} \text{ (By definition of } x_2) \\
&= \frac{m - (m-1)x_1}{2x_2} \lceil D/T \rceil \text{vol}(G)
\end{aligned}$$

As we have observed earlier in this proof, during the interval I EDF is only processing jobs with deadlines in I. The cumulative execution requirement of these jobs is bounded from above by $\lceil D/T \rceil \text{vol}(G)$. For a deadline miss to occur, it is therefore necessary that

$$\begin{aligned}
& \frac{m - (m-1)x_1}{2x_2} < 1 \\
\Leftrightarrow \quad & m - (m-1)x_1 < 2x_2
\end{aligned}$$

By negating the condition above and rearranging terms, we conclude that

$$m \geq (m-1)x_1 + 2x_2$$

is a sufficient condition for the sporadic DAG to be EDF-schedulable on m unit-speed processors, as claimed in the lemma. □

Theorem 1 suggests the following polynomial-time sufficient EDF-schedulability test:

EDFSCHED$\Big((G, D, T), m\Big)$
  compute $\text{len}(G)$
  compute $\text{vol}(G)$
  if Condition (2) holds **return** (schedulable)
  return (not known to be schedulable)

A. Determining the number of processors

Recall from Section I that we had discussed the problem of implementing recurrent tasks such as sporadic DAG tasks upon multicore platforms with a very large number of processors, upon which some processors are dedicated to the exclusive use of particular tasks. One important resource-allocation question that arises in such platforms is that of determining how many processors are to be assigned to a given task, in order to ensure that it meet its deadlines. Theorem 1 can be used for answering this question for EDF-scheduled systems: Given a sporadic DAG task (G, D, T), we can compute $\text{len}(G)$ and $\text{vol}(G)$, and then seek the smallest m satisfying Condition (2). Algebraic simplification of Condition (2) reveals that this smallest number of processors is equal to

$$\left\lceil \frac{(2\text{vol}(G)/T) - (\text{len}(G)/D)}{1 - (\text{len}(G)/D)} \right\rceil. \tag{3}$$

B. A speedup bound

We now derive a speedup bound for the EDF-schedulability test EDFSCHED. Recall from Definition 1 that a schedulability test is said to have a speedup bound b if any task that can be scheduled by an optimal algorithm on m unit-speed processors is determined schedulable by the test on m speed-b processors. We will prove (Theorem 2 below) that EDFSCHED has a speedup bound of $(3 - 1/m)$.

First, we establish two necessary conditions for G to be feasible upon a platform comprised of speed-x processors.

Lemma 2. If (G, D, T) is feasible on m speed-x processors, then

1) $\text{len}(G) \leq xD$, and
2) $\text{vol}(G) \leq xmT$.

Proof: If $\text{len}(G) > xD$, then the longest chain of each dag-job released by the task will fail to meet its deadline on speed-x processors if each job on this chain needs to execute for its WCET.

If $\text{vol}(G) > mxT$, then the total execution requirement of the synchronous arrival sequence would eventually exceed the available computing capacity of the platform. □

Lemma 2 defines necessary conditions for a sporadic DAG task to be feasible; we will now use Theorem 1 to show that

these conditions become sufficient with a speedup of $(3 - 1/m)$:

Lemma 3. Any sporadic DAG (G, D, T) that is feasible on m speed-$(\frac{m}{3m-1})$ processors is determined to be EDF-schedulable on m unit-speed processors by Algorithm EDFSCHED.

Proof: Suppose that (G, D, T) is s feasible on m speed-$(\frac{m}{3m-1})$ processors. By Lemma 2 above with $x \leftarrow (\frac{m}{3m-1})$, it must be the case that $\text{len}(G) \leq (\frac{m}{3m-1})D$ and $\text{vol}(G) \leq (\frac{m}{3m-1})mT$. We will use these inequalities to evaluate the LHS of Condition (2).

$$\begin{aligned}
\text{LHS} &= (m-1)\frac{\text{len}(G)}{D} + 2\frac{\text{vol}(G)}{T} \\
&\leq (m-1)\left(\frac{m}{3m-1}\right) + 2m\left(\frac{m}{3m-1}\right) \\
&= \frac{m^2 - m + 2m^2}{3m-1} \\
&= \frac{m(3m-1)}{3m-1} \\
&= m = \text{RHS}
\end{aligned}$$

and the lemma is proved. □

Observing that $(3 - 1/m)$ is the multiplicative inverse of $\frac{m}{3m-1}$, we conclude

Theorem 2. The EDF-schedulability test EDFSCHED has a speedup bound of $(3 - 1/m)$.

C. An improvement

We now derive an additional result that can sometimes improve upon the performance of Algorithm EDFSCHED, in the sense that this result allows us to determine the EDF-schedulability of some sporadic DAG tasks that may not be deemed schedulable by Algorithm EDFSCHED. We will also see (Theorem 4) that incorporating this improvement into the schedulability test results in a test with a superior (i.e., smaller) speedup bound.

Theorem 3. Any sporadic DAG task (G, D, T) satisfying the following properties

1) $\text{len}(G) \leq 2D/5$
2) $\text{vol}(G) \leq 2mT/5$

is EDF-schedulable on m unit-speed processors.

Proof: Suppose for contradiction that EDF fails to meet all deadlines while scheduling some sequence of dag-jobs released by the sporadic DAG. Let j be the first job that misses its deadline d_j. W.l.o.g. we assume that there are no jobs with a deadline later than d_j. Consider the interval $I := [r_j, d_j)$. Denote by X the total amount of time during I where all processors are busy. Let $Y := (d_j - r_j) - X$, i.e., Y denotes the total amount of time during I where not all processors are busy.

We first observe that $Y \leq \frac{2}{5}D$. This follows from the observation that whenever a processor is idle EDF must be executing a job belonging the longest chain of the last activation of G and hence $Y \leq \text{len}(G)$, which is assumed $\leq \frac{2}{5}D$.

Condition $Y \leq \frac{2}{5}D$ implies that $X \geq \frac{3}{5}D$. Now since the total amount of execution occurring over the interval I is $\geq (mX + Y)$, we conclude that

$$\text{The total work done by EDF during } I \text{ is } \geq \frac{3m}{5}D. \quad (4)$$

We now distinguish between two cases: first, when $D \geq 2T$ and next when $D < 2T$.

§**1.** First we assume that $D \geq 2T$ and notice that in this case $\lceil \frac{D}{T} \rceil \leq \frac{3}{2} \cdot \frac{D}{T}$. Therefore the total amount of execution that could be required during the interval I is bounded from above by

$$\begin{aligned}
\left\lceil \frac{D}{T} \right\rceil \text{vol}(G) &\leq \frac{3}{2}\frac{D}{T}\text{vol}(G) \\
&\leq \frac{3}{2}\frac{D}{T} \times \frac{2}{5}\,m\,T \text{ (by assumption 2)} \\
&\leq \frac{3}{5}D\,m
\end{aligned}$$

By (4) above, this is no larger than the amount of work done during I; this contradicts the assumption that EDF fails.

§**2.** In the second case, we assume $D < 2T$. Let us define $\gamma < 1$ such that $D = (1 + \gamma)T$; we identify two sub-cases:
§**2.1.** First assume that $\gamma \geq \frac{1}{2}$ and observe that in such a case

$$\left\lceil \frac{D}{T} \right\rceil \leq (2 - \gamma)\frac{D}{T}.$$

Then the total execution that could be required during the interval I is bounded from above by

$$\begin{aligned}
\left\lceil \frac{D}{T} \right\rceil \text{vol}(G) &\leq (2-\gamma)\frac{D}{T}\text{vol}(G) \\
&\leq (2-\gamma)\frac{D}{T} \times \frac{2}{5}\,m\,T \text{ (by assumption 2)} \\
&\leq \frac{3}{2} \times \frac{2}{5}\,D\,m \text{ (by definition of } \gamma) \\
&\leq \frac{3}{5}D\,m
\end{aligned}$$

which, in conjunction with condition (4) above, again contradicts the assumption that EDF fails.

§**2.2.** Finally, assume that $D < 2T$ and $\gamma < 1/2$. In this case, we complete at most $\gamma T m$ units of execution during the interval $[r_j, r_j + \gamma T)$; during the interval $[r_j + \gamma T, d_j)$ EDF completes less than $\text{vol}(G)$ units of execution (since only the last dag-job is available during this interval, and we are assuming that EDF fails). Hence, the total work processed is bounded by

$$\begin{aligned}
\gamma\,m\,T + vol(G) &\leq \gamma\,m\,T + \frac{2}{5}\,m\,T \text{ (by assumption 2)} \\
&= \gamma\,m\,\frac{D}{1+\gamma} + \frac{2}{5}\,m\,\frac{D}{1+\gamma} \text{ (by definition of } \gamma) \\
&= m\,D\left(\frac{\gamma + \frac{2}{5}}{1+\gamma}\right) \\
&\leq m\,D\,\frac{3}{5}
\end{aligned}$$

EDFSCHED$\Big((G, D, T), m\Big)$
1 compute $\text{len}(G)$
2 compute $\text{vol}(G)$
3 **if** $\Big(\text{len}(G) \leq \frac{2D}{5}\Big)$ and $\Big(\text{vol}(G) \leq \frac{2mT}{5}\Big)$ **return** (schedulable)
4 **if** Condition (2) holds **return** (schedulable)
5 **return** (not known to be schedulable)

Fig. 2. The improved sufficient EDF-schedulability test.

where the last step follows from the observation that since the function $f(\gamma) = \frac{\gamma+\frac{2}{5}}{1+\gamma}$ is monotonically increasing, its value within the interval $[0, \frac{1}{2})$ is upper bounded by $f(\frac{1}{2}) = \frac{\frac{1}{2}+\frac{2}{5}}{1+\frac{1}{2}} = \frac{3}{5}$. This, taken in conjunction with condition (4), again contradicts the assumption that EDF fails. □

The result of Theorem 3 can be incorporated into Algorithm EDFSCHED — the improved algorithm is listed in Figure 4. Lines 1, 2, 4, and 5 of the pseudo-code are unchanged from the previous version; Line 3 incorporates the result derived in Theorem 3.

By combining Lemma 2 with Theorem 3 and observing that $\frac{5}{2} = 2.5$ is the multiplicative inverse of $\frac{2}{5}$, we have the following improved speedup bound.

Theorem 4. The EDF- schedulability test EDFSCHED of Figure 2 has a speedup bound of 2.5.

VII. EDF SCHEDULABILITY TESTING IN PSEUDO-POLYNOMIAL TIME

In this section we prove (Theorem 5) that the EDF scheduling algorithm has a speedup bound no larger than 2 for scheduling sporadic DAG tasks. This proof introduces a novel notion of the computational *demand* and *load* of a sporadic DAG task. We subsequently build upon these ideas to obtain a pseudo-polynomial time schedulability test for EDF – this test is presented in Figure 4.

A. Defining a Demand Bound Function and Load

We will now define a novel demand function that, for a given collection of dag-jobs, captures the amount of work due in each time interval. This is accomplished in a manner that accounts for the precedence constraints.

Preprocessing: Reduction to unit WCETs. Given sporadic DAG task (G, D, T), we replace each vertex of G with WCET $p_v > 1$ by a chain of p_v vertices each having unit WCET. It is evident that this transformation takes pseudo-polynomial time.

Figure 3 shows the result of performing these preprocessing steps on the DAG of Figure 1. The vertex j_3, with WCET 2, is split into a chain of two vertices. There are four layers numbered 0, 1, 2, and 3.

Preprocessing: Organization into layers. We then structure G in *layers*. Each node $v \in V$ with no predecessor is assigned to layer $\ell(v) = 0$. For each other node v, we inductively define its layer as $\ell(v) = 1 + \max_{u \in V:(u,v)\in A} \ell(u)$. Let $V_i = \{v \in V : \ell(v) = i\}$.

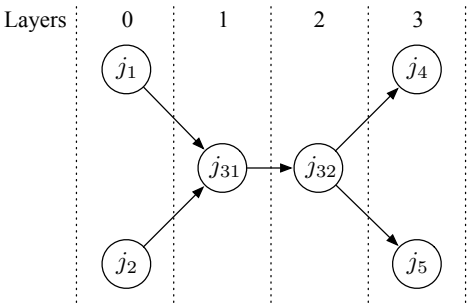


Fig. 3. Pre-processing the sporadic DAG task of Figure 1. Each vertex has a WCET of 1. The vertex layers are labeled.

The layers have the property that if dag-job j is released at time r_j, then the job corresponding to node v of j cannot be started before time $r_j + \ell(v)$ by any unit speed scheduling algorithm, independently of the number of available processors. The quantity $r_j + \ell(v)$ is called the *shifted release time* of the job. The transformation is similar in spirit to one used by Saifullah et al. [11].

Definition 2 (**Shifted release times demand**). *For a collection of dag-jobs R and an interval $[a, b]$, the shifted release times demand, $\text{SD}_R[a, b]$, is the cumulative execution requirement of the unit-sized jobs j of the dag-jobs in R that have both their deadlines and their shifted release times within the interval $[a, b]$.*

Observe how the quantity $\text{SD}_R[a, b]$ captures the contribution of some jobs whose actual release time is strictly before a. Therefore, it is a proper refinement of the standard notion of demand.

Definition 3 (**Shifted release times demand bound function**). *For any $L \in \mathbb{N}$, the shifted release times demand bound function is defined as follows*

$$\text{SDBF}(L) := \sup_{R,a} \Big(\text{SD}_R[a, a + L]\Big), \tag{5}$$

where R ranges over all sequences of dag-jobs that can legally be generated by the DAG task (G, D, T).

Definition 4 (**The load λ**). *Define the* load λ *of a sporadic DAG task to be*

$$\lambda := \sup_{L} \frac{\text{SDBF}(L)}{L} \tag{6}$$

In Section VII-D we will describe how a task's load parameter λ can be determined in time pseudo-polynomial in the representation of the task; the significance of this parameter arises from the following (straightforward) result:

Lemma 4. If a sporadic DAG task is feasible on m unit speed processors, then $\lambda \leq m$ for this task.

Proof: Suppose $\lambda > m$, and consider some L for which $\text{SDBF}(L) > Lm$. Consider a sequence of dag-jobs R and a time a defining $\text{SDBF}(L)$ for this value of L; mL is a

lower bound on the amount of execution that any scheduling algorithm needs to complete within the interval $[a, a+L]$, when scheduling the sequence of dag-jobs R. The sequence of dag-jobs R cannot therefore be successfully scheduled on m unit-speed processors. □

B. A speedup bound for EDF

In Lemma 5 below, we present a sufficient EDF-schedulability condition for a sporadic DAG task (D, G, T) on speed-2 processors, given the parameters λ and $\text{len}(G)$ for the task. We will then use this result to prove that EDF has a speedup bound no larger than two for scheduling sporadic DAG tasks.

Lemma 5. Any sporadic DAG task (G, D, T) with load $\lambda \leq m$ and $\text{len}(G) \leq D$ is successfully scheduled by EDF on m speed-2 processors.

Proof: Suppose that sporadic DAG task (G, D, T) has $\lambda \leq m$ and $\text{len}(G) \leq D$, and EDF fails to schedule some dag-job sequence of this task. Let R denote a minimal dag-job sequence on which EDF misses a deadline. Let j denote a job which misses its deadline d_j. Since R is minimal, there are no jobs in R with a deadline later than d_j. Denote by t^* the latest instant strictly before d_j such that all pending jobs at time t^* have shifted release time $\geq t^*$. (Such an instant must exist, since the time-instant 0 satisfies this property.) Since EDF runs with speed 2, we can view the interval $[t^*, d_j)$ as being composed of "half-slots" each of duration $1/2$ time units. We partition this interval into maximal intervals $X_1, ..., X_k$ and $Y_1, ..., Y_k$ such that each interval X_j has the property that during each half-slot $t \in X_j$ all processors are busy, and correspondingly each interval Y_j has the property that at each half-slot $t \in Y_j$ at least one processor is idled. We define $X := \sum_i |X_i|$ and $Y := \sum_i |Y_i|$. We consider separately the two cases: $X \geq Y$, and $X < Y$. For each case, we will derive a contradiction, thereby showing that it is not possible that EDF fails to schedule R.

Case $X \geq Y$**.** In this case, $X \geq (d_j - t^*)/2$. Since EDF runs with speed 2 processors, the total amount of execution completed by EDF on the speed-2 processors during the interval $[t^*, d_j]$ must be strictly more than $m(d_j - t^*)$ (note here that $d_j - t^* > 0$). But by the definition of t^*, during $[t^*, d_j]$ EDF processes only jobs that contribute to $\text{SD}_R[t^*, d_j]$. But since

$$\begin{aligned} \text{SD}_R[t^*, d_j] &\leq \text{SDBF}(d_j - t^*) \text{ (By definition of SDBF)} \\ &\leq \lambda \times (d_j - t^*) \quad \text{(By definition of load } \lambda) \\ &\leq m(d_j - t^*) \end{aligned}$$

we obtain a contradiction.

Case $X < Y$**.** The crucial observation here is that during each half-slot in Y_j (for any interval Y_j) the algorithm completes the pending jobs[1] of at least one layer of each currently pending (and in particular unfinished) dag-job – otherwise the processors would all be busy during Y_j. Let t' denote the earliest instant after t^* such that $\sum_j |Y_j \cap [t^*, t')| > \sum_j |X_j \cap [t^*, t')|$. That is, t' is the earliest instant $> t^*$ such that over $[t^*, t')$, there are strictly more half-slots with at least one processor idled than there are with all processors busy. (Such a t' is guaranteed to exist since we are assuming that $Y > X$.) Observe that $t' \leq d_j$ since otherwise we would have a contradiction to the minimality of t'.

Observe that the interval $[t^*, t')$ contains $2(t' - t^*)$ half-slots. Of these, more than half — i.e., $> (t' - t^*)$ — have at least one processor idled (by definition of t'). As noted above, EDF completes the pending jobs of at least one layer of each currently pending dag-job. Since there can be no more than $(t' - t^*)$ layers with shifted release times in $[t^*, t')$, this implies that at t' we would have completed all work on all pending jobs except the ones with shifted release time t' or larger. If $t' < d_j$ then this contradicts the definition of t^* as the *latest* point in time strictly before d_j with that property. If $t' = d_j$ then all pending jobs have a shifted release date of d_j or larger and by assumption there is at least one pending job. But this is only possible if $\text{len}(G) > D$, contradicting our assumption. □

[1]Recall that after our preprocessing step, each such job has WCET equal to 1.

EDFSCHEDPP$\Big((G, D, T), m\Big)$

1 obtain G' from G, by doubling each vertex's WCET
2 preprocess G' to have unit WCET's
3 compute $\text{len}(G')$
4 **if** $(\text{len}(G') > D)$ **return** (not known to be schedulable)
5 compute λ for (G', D, T) as described in Section VII-D
6 **if** $(\lambda > m)$ **return** (not known to be schedulable)
7 **return** (schedulable)

Fig. 4. The pseudo-polynomial-time sufficient EDF-schedulability test.

As a direct consequence of Lemma 5 and Lemma 4 we obtain

Theorem 5. EDF has a speedup bound no larger than two for scheduling sporadic DAG tasks.

Proof: Suppose that sporadic DAG task (G, D, T) is feasible on m unit-speed processors. We conclude, based on Lemma 4 above, that $\lambda \leq m$ for this task. We also note that $\text{len}(G)$ is necessarily $\leq D$. Hence it follows from Lemma 5 that (D, G, T) is EDF-schedulable on m speed-2 processors. □

C. A pseudo-polynomial time EDF schedulability test

We now describe how to use the result of Lemma 5 to obtain a sufficient EDF-schedulability condition on unit-speed processors. The basic idea is to compute $\text{len}(G)$ and λ *assuming that the task were executing on a platform comprised of speed-*0.5 *processors*; if the $\text{len}(G)$ and λ so computed satisfy the conditions that $\text{len}(G) \leq D$ and $\lambda \leq m$, then Lemma 5 allows us to conclude that it is EDF-schedulable on m unit-speed processors.

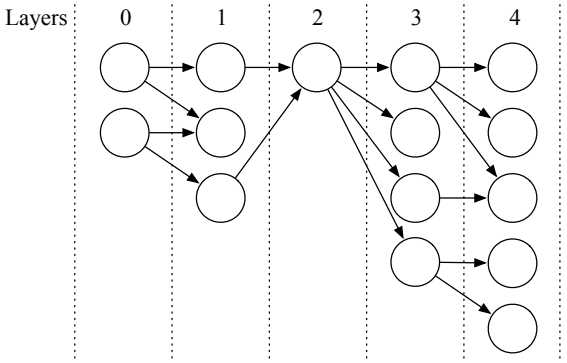


Fig. 5. Another sporadic DAG task ($D = 7$, $T = 2$). The DAG has already been pre-processed to have unit WCETs.

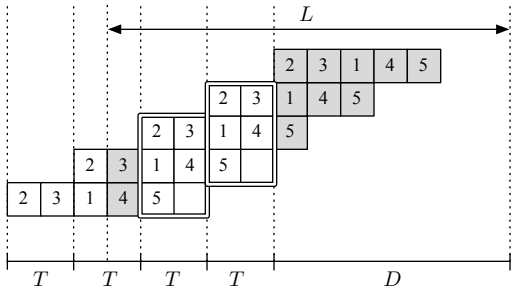


Fig. 6. Computation of $\text{SDBF}(L)$ for the DAG task of Figure 5. Each row of the drawing corresponds to a dag-job; time-slots are labeled with the demand of the corresponding layer of the graph. Each doubly-framed area corresponds to a total demand $\text{vol}(G)$; the shaded area represents $f(G, r)$. In this example, $D = 7$, $T = 2$, $L = 12$, $q = 2$, $r = 1$.

The crucial observation is that since a job takes twice as long to execute upon a speed-0.5 processor as on a unit-speed one, computing $\text{len}(G)$ and λ assuming execution on speed-0.5 processors is equivalent to computing $\text{len}(G')$ and λ on unit-speed processors of the sporadic DAG (G', D, T), in which G' is obtained from G by doubling the WCET that labels each vertex of G. This observation leads to the sufficient EDF-schedulability test in Figure 4. We have already observed that the preprocessing (Line 2) takes pseudo-polynomial time; we will see in Section VII-D below that determining λ (Line 5) can also be done in time pseudo-polynomial in the representation of the DAG task. It is evident that the remaining steps take polynomial time, allowing us to conclude that Algorithm EDFSCHEDPP has pseudo-polynomial run-time.

We note that Algorithm EDFSCHEDPP can also be used to answer the question considered in Section VI-A: how many processors must be assigned to a given task, in order to ensure that it meet its deadlines? If a task fails the test of Line 4, then we cannot schedule it to meet all deadlines; else, we can assign it $\lceil \lambda \rceil$ processors.

D. Computing the load

In this section we describe how we may determine the load λ for a given sporadic DAG task (G, D, T) in time pseudo-polynomial in the representation of the task. But first, we obtain a characterization of the collections of dag-jobs that define the shifted release times demand bound functions (the $\text{SDBF}(L)$ values):

Lemma 6. For any $L \geq 0$, $\text{SDBF}(L)$ is equal to $\text{SD}_R[a, a+L]$ for values of R and a satisfying the following two properties:

- dag-jobs in R are released as soon as possible (i.e., exactly T time units apart); and
- the time-instant $a + L$ is the deadline of some dag-job in R.

Proof: These can be validated by observing that for any a, R not satisfying these properties, the total computational demand over an interval of length L can only *in*crease by first moving the interval $[a, a + L]$ leftwards (i.e., backwards in time) until the right end of the interval coincides with a deadline in R, and then moving dag-job arrivals rightwards (forward in time) as much as possible subject to the period constraint. □

As a consequence of Lemma 6, $\text{SDBF}(L)$ can be computed in time polynomial in L, by simply considering a sequence of dag-jobs that arrive as soon as possible and that together span an interval of length no more than $(L + T)$.

§**1.** We first describe how to determine $\sup_{L<D}(\text{SDBF}(L)/L)$. For pseudo-polynomial values of L, Lemma 6 implies that we can determine $\text{SDBF}(L)/L$ exactly in pseudo-polynomial time. Furthermore, since there are only pseudo-polynomially many distinct values of L that are $\leq D$, we can determine

$$\sup_{L \leq D} \frac{\text{SDBF}(L)}{L}$$

in pseudo-polynomial time by determining the ratio $\text{SDBF}(L)/L$ separately for each each value of $L \leq D$.

§**2.** We now describe how to determine $\sup_{L \geq D}(\text{SDBF}(L)/L)$.

§**2.1.** Let r denote any integer $0 \leq r \leq T-1$. Consider all $L \geq D$ that satisfy $(L-D) \bmod T \equiv r$. Any such L can be written as $L = D + qT + r$, for some $q \in \mathbb{N}$. Considering only such values of L, determining $\sup_L(\text{SDBF}(L)/L)$ is equivalent to finding

$$\sup_{q \in \mathbb{N}} \left(\frac{W}{qT + D + r} \right),$$

where W is the maximum number of jobs having shifted release time and deadline in an interval of length $qT + D + r$. The key observation is that $W = q \cdot \text{vol}(G) + f(G, r)$, where $f(G, r)$ is a quantity that depends only on the graph G and r, not on q (see Figures 5 and 6 for an illustration). We can also conclude, using an argument essentially identical to that used in the proof of Lemma 6, that $f(G, r)$ can be computed in time polynomial in $(D+r)$. Hence determining $\sup_L(\text{SDBF}(L)/L)$ for all these values of L is equivalent to determining

$$\sup_{q \in \mathbb{N}} \frac{q\,\text{vol}(G) + f(G, r)}{q\,T + D + r}.$$

Since the right hand side is monotonic in q, the supremum equals

$$\max\left(\frac{\mathrm{vol}(G)}{T}, \frac{f(G,r)}{D+r}\right).$$

§**2.2.** Above, we described how we would determine $\sup_L(\textsc{sDBF}(L)/L)$ for all these values of $L \geq D$ that can be expressed as $L = D + qT + r$, for a given r. We now repeat the above process for each $r \in \mathbb{N}$, $0 \leq r < T$; since there are only pseudo-polynomially many such r, this takes pseudo-polynomial time.

§**3.** The largest value from among those obtained in steps §1 and §2 above is the value of λ; since each step above takes pseudo-polynomial time and there are pseudo-polynomially many steps, we have shown that λ can be determined in time pseudo-polynomial in the representation of the sporadic DAG task (G, D, T).

E. Combining the tests

In this paper, we have derived polynomial and pseudo-polynomial sufficient schedulability tests for determining whether a sporadic DAG task is EDF schedulable on an identical multiprocessor platform. We now explain how we envision these tests being used in concert. Given a sporadic DAG task (G, D, T) and a number m of unit-speed processors, we would first compute, in polynomial time, the quantities $\mathrm{vol}(G)$ and $\mathrm{len}(G)$. If $\mathrm{len}(G) > 1$ or $\mathrm{vol}(G) > m$, we declare the task non-schedulable, while if $\mathrm{len}(G) \leq 2/5$ and $\mathrm{vol}(G) \leq (2m)/5$, we declare it EDF-schedulable. Else, we determine, again in polynomial time, whether Condition (2) is satisfied; if so, we declare the task EDF-schedulable. Failing this, we finally call EDFSCHEDPP (Figure 4). In this manner, the computationally most expensive (pseudo-polynomial time) test is only called when the polynomial-time test fails to return an authoritative answer.

VIII. Summary and Conclusions

Recent advances in computer architecture and fabrication technology have made it possible to build multi-core processors with perhaps *thousands* of computing cores in a single CPU. The presence of large core-counts offers new opportunities for executing more computation-intensive workloads in real time. However, in order to better exploit multi-core computing capabilities it is necessary that more expressive formal models that are more capable of exposing the parallelism within these workloads be used. In this paper, we have studied one such model in some detail. This model combines the *DAG-based* models that have been widely studied in the "traditional" scheduling (Algorithms, and Operations Research) communities, with the *recurrent* models used in real-time systems. By allowing for the expression of both *dependencies* and *parallelism* within the workload, as well as for representing the recurrent nature of many real-time processes, such a recurrent DAG task model seems particularly appropriate for modeling real-time workloads on multiprocessor platforms.

If the deadline parameter of a recurrent DAG task is no larger than its period, then at most one dag-job may be active at any given instant in time. For such tasks, we were able to exploit prior results from traditional scheduling theory to obtain efficient (i.e., polynomial-time) scheduling algorithms and schedulability tests that have good speedup bounds. If the deadline parameter may exceed the period, however, prior techniques are no longer adequate. We have introduced a series of novel concepts and techniques for dealing with such tasks, and have applied these concepts and techniques to the analysis of sporadic-DAG tasks that are scheduled using EDF. We are hopeful that some of these techniques will be useful in solving scheduling problems for parallel recurrent workload models more general than the one considered in this paper.

We end by listing a couple of generalizations that seems particularly pertinent:

- Each vertex may have a different relative deadline parameter and/or a release time specified.
- There may be multiple sporadic DAG tasks sharing a given collection of processors.

Acknowledgment

The authors would like to thank Enrico Bini for bringing the problem studied in this paper to their attention.

References

[1] V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "A constant-approximate feasibility test for multiprocessor real-time scheduling," *Algorithmica*, vol. 62, no. 3–4, pp. 1034–1049, 2012.

[2] M. Dertouzos, "Control robotics : the procedural control of physical processors," in *Proceedings of the IFIP Congress*, 1974, pp. 807–813.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

[4] R. Graham, "Bounds on multiprocessor timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416–429, 1969.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[6] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *RTSS*. IEEE Computer Society, 2010, pp. 259–268.

[7] J. K. Lenstra and A. H. G. Rinnooy Kan, "Complexity of scheduling under precedence constraints," *Operations Research*, vol. 26, no. 1, pp. 22–35, 1978.

[8] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[9] C. L. Liu, "Scheduling algorithms for hard real-time programming of a single processor," *JPL Space Programs Summary 37-60*, vol. II, pp. 31–37, 1969.

[10] ——, "Scheduling algorithms for multiprocessors in a hard real-time environment," *JPL Space Programs Summary 37-60*, vol. II, pp. 28–31, 1969.

[11] A. Saifullah, K. Agrawal, C. Lu, and C. D. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *Proc. IEEE Real-Time Systems Symposium*, 2011, pp. 217–226.

[12] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. Chicago: IEEE Computer Society Press, 2011, pp. 71–80.

[13] ——, "On the tractability of digraph-based task models," in *Proceedings of the EuroMicro Conference on Real-Time Systems*. Porto, PT.: IEEE Computer Society Press, July 2011.

[14] O. Svensson, "Hardness of precedence constrained scheduling on identical machines," *SIAM Journal on Computing*, vol. 40, no. 5, pp. 1258–1274, 2011.

[15] J. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384 – 393, 1975.