The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks

Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela

The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks

Sanjoy Baruah
The University of North Carolina
USA
baruah@cs.unc.edu

Vincenzo Bonifaci
IASI–CNR
Italy
vincenzo.bonifaci@iasi.cnr.it

Alberto Marchetti-Spaccamela
Sapienza Università di Roma
Italy
alberto@dis.uniroma1.it

***Abstract*—The sporadic DAG task model exposes parallelism that may exist within individual tasks to the run-time scheduling mechanism, and is therefore considered a particularly suitable model for representing recurrent real-time tasks that are to be implemented upon multiprocessor platforms. This paper proposes and evaluates an extension to the model to allow for the concurrent modeling of conditional execution of pieces of an individual task, along with the modeling of intra-task parallelism. The Global Earliest Deadline First (GEDF) scheduling of systems represented in this generalized model is studied, and a GEDF-schedulability test is derived. With regards to GEDF scheduling it is shown that there is no penalty, in terms of a worse speedup factor, in generalizing the sporadic DAG tasks model in this manner.**

I. Introduction

The *sporadic DAG task* model [3] was introduced to permit the representation of parallelism that may be present within individual recurrent tasks. A task τ_i in this model is specified as a 3-tuple (G_i, D_i, T_i), where G_i is a directed acyclic graph (DAG), and D_i and T_i are positive integers representing the relative deadline and period parameters of τ_i respectively. The task τ_i repeatedly releases *dag-jobs*, each of which is a collection of (sequential) jobs. Successive dag-jobs are released a duration of at least T_i time units apart. The DAG G_i is specified as $G_i = (V_i, E_i)$, where V_i is a set of vertices and E_i a set of directed edges between these vertices. Each $v \in V_i$ represents the execution of a sequential piece of code (such execution is called a "job"), and is characterized by a worst-case execution time (wcet). The edges represent dependencies between the jobs: if $(v_1, v_2) \in E_i$ then job v_1 must complete execution before job v_2 can begin execution. (Job v_1 is called a *predecessor* job of v_2, and job v_2 is called a *successor* job of v_1.) Jobs that are not predecessors or successors of each other, either directly or transitively, may execute simultaneously upon different processors. A release of a dag-job of τ_i at time-instant t means that all $|V_i|$ jobs $v \in V_i$ are released at time-instant t. If a dag-job is released at time-instant t then all $|V_i|$ jobs that were released at t must complete execution by time-instant $t + D_i$.

As stated above, the sporadic DAG tasks model assumes that each release of a dag-job of τ_i causes the release of jobs corresponding to each and every vertex in V_i. However, control structures (such as conditional — *if-then-else* —constructs) within the code that is being modeled by the task may mean that different activations of the task (i.e., different dag-jobs) cause different parts of the code to be executed. Assuming that jobs corresponding to all the vertices in V_i will execute during each such activation is pessimistic; there is a need to be able to model the fact that different dag-jobs of the same task may cause different collections of jobs to be executed. To our knowledge, the *multi-DAG model* proposed by Fonseca et al. [6] represents the first attempt at concurrently modeling both intra-task parallelism and conditional execution in recurrent real-time task systems. The multi-DAG model models each recurrent task as a collection of "execution flows," each of which represents a different flow of control through the code being modeled by the task; each such execution flow is explicitly modeled as a separate DAG.

Although the multi-DAG model does indeed succeed in achieving its goal of generalizing the sporadic DAG task model to represent conditional control-flow constructs, this generalization comes at a significant price in terms of computational complexity. As stated above, each possible flow of control (called "execution flow") through the code modeled by an individual task is explicitly represented by a separate DAG, and the *number* of such flows is an important parameter in determining the efficiency of the schedulability analysis and run-time scheduling algorithms proposed in [6]. But there may in general be exponentially many different flows through a graph. Consider for example code structured like this:

```
if (C1) then {S11} else {S12}
if (C2) then {S21} else {S22}
if (C3) then {S31} else {S32}
. . . . . .
. . . . . .
if (Cn) then {Sn1} else {Sn2}
```

where each `(Ci)` represents a boolean condition, and each `{Sij}` a block of straight-line code. It is evident that such a code fragment may have 2^n different execution flows through it; hence, requiring explicit enumeration of all execution flows and having the number of such flows be a determinant in the computational complexity of scheduling and schedulability analysis algorithms means that these algorithms all have exponential worst-case run-time.

Other related work. The scheduling problem of parallel tasks on multiprocessor systems is receiving significant attention in the RT community. Parallel task models have been proposed to represent the task parallelism required by paralleling programming models (e.g. OpenMP): we mention the fork-join model [7] and the synchronous parallel model [12], [1], [10], [5] that are more restrictive than the DAG model. Regarding the DAG model, several results have been

recently obtained about schedulability tests for EDF and other scheduling policies [3], [4], [8], [11], [2], [13], [9]. To the best of our knowledge, all previous research on the DAG model assumes that there are no conditional statements.

This research. We propose the *conditional sporadic DAG task* model as an extension to the sporadic DAG task model [3] that is capable of modeling certain conditional control-flow constructs (including the cascade of `if-then-else` commands depicted above). We consider the Global Earliest-Deadline First (GEDF) scheduling of task systems that are modeled as collections of conditional sporadic DAG tasks; this is in contrast to the approach of [6], which develops entirely new (and rather complicated) server-based mechanisms for the run-time scheduling of systems of multi-DAG tasks. We quantitatively evaluate the effectiveness of GEDF as a scheduling mechanism for systems of conditional sporadic DAG tasks via the speedup factor metric. We show that the tight speedup bound of $\left(2-\frac{1}{m}\right)$ that was obtained in [4] for the GEDF scheduling of traditional (non-conditional) sporadic DAG task systems is easily shown to hold for systems of conditional sporadic DAG tasks as well. This means that at least from the perspective of speedup factor, the added expressive capabilities of the conditional sporadic DAG tasks model comes at no additional cost.

Deriving an effective GEDF schedulability test for conditional sporadic DAG task systems turns out to be more challenging than showing the speedup bound – straight-forward extensions of the techniques from [4] require the enumeration of all paths through the DAG-representation of the task (as in the approach of [6]), and result in exponential-time algorithms. We develop a novel transformation strategy that converts each conditional sporadic DAG task to a non-conditional one in polynomial time, and tests the system of transformed tasks for GEDF schedulability using the test provided in [4]. We show that the resulting GEDF schedulability test for conditional sporadic has a speedup factor equal to $(2 - 1/m + \epsilon)$ for any constant $\epsilon > 0$; once again, this is the same result as was available for traditional (i.e., not conditional) sporadic DAG task systems.

Organization. The remainder of this paper is organized as follows. In Section II we formally define the conditional sporadic DAG tasks model, a generalization of the sporadic DAG tasks model of [3]. In Section III we review some prior work, primarily from [4], on the GEDF scheduling of task systems represented using the traditional (non-conditional) sporadic DAG tasks model. In Sections IV and V we describe how the results described in Section III may be extended to hold for task systems that are represented using the conditional sporadic DAG tasks model as well.

II. Conditional Sporadic DAG tasks

Conditional sporadic DAG tasks are a generalization to the traditional sporadic DAG tasks [3] as described in Section I above. As with the traditional sporadic tasks, each conditional sporadic DAG task τ_i is specified as a 3-tuple (G_i, D_i, T_i), where $G_i = (V_i, E_i)$ is a DAG, and D_i and T_i are positive integers. We require that G_i have a single source vertex and a single sink vertex.[1] *Conditional vertices* are special vertices in V_i that are defined in pairs. Let (c_1, c_2) be such a pair in the DAG $G_i = (V_i, E_i)$. Informally speaking, vertex c_1 can be thought of representing a point in the code where a conditional expression is evaluated and, depending upon the outcome of this evaluation, control will subsequently flow along exactly one of several different possible paths in the code. It is required that all these different paths meet again at a common point in the code, represented by the vertex c_2. More formally,

1) There are multiple outgoing edges from c_1 in E_i. Suppose that there are exactly k outgoing edges from c_1 to the vertices $s_1, s_2, \ldots, s_k$, for some $k > 1$. We call k the *branching factor* of this conditional. Then there are exactly k incoming edges into c_2 in E_i, from the vertices $t_1, t_2, \ldots, t_k$,
2) For each $\ell \in \{1, 2, \ldots, k\}$, let $V'_\ell \subseteq V_i$ and $E'_\ell \subseteq E_i$ denote all the vertices and edges on paths reachable from s_ℓ that do not include vertex c_2. By definition, s_ℓ is the sole source vertex of the DAG $G'_\ell \stackrel{\text{def}}{=} (V'_\ell, E'_\ell)$. It must hold that t_ℓ is the sole sink vertex of G'_ℓ.
3) It must hold that $V'_\ell \bigcap V'_j = \emptyset$ for all $\ell, j, \ell \neq j$. Additionally, with the exception of (c_1, s_ℓ) there should be no edges in E_i into vertices in V'_ℓ from vertices not in V'_ℓ, for each $\ell \in \{1, 2, \ldots, k\}$. I.e., $E_i \bigcap((V_i \setminus V'_\ell) \times V'_\ell) = \{(c_1, s_\ell)\}$ should hold for all ℓ.

Edges (v_1, v_2) between pairs of vertices neither of which are conditional vertices represent precedence constraints exactly as in traditional sporadic DAG tasks, while edges involving conditional vertices represent conditional execution of code. More specifically, let (c_1, c_2) denote a defined pair of conditional vertices (recall that conditional vertices are always defined in pairs).

- After the job c_1 completes execution, exactly one of its successor jobs becomes eligible to execute; it is not known beforehand which successor job may execute.
- Job c_2 begins to execute upon the completion of exactly one of its predecessor jobs.

The branching factor for an "if-then-else" condition is 2. Without loss of generality, we will assume in this paper that all conditionals have branching factor 2; conditionals with branching factor > 2 can always be converted to an equivalent sequence of conditionals with branching factor two with no more than a polynomial increase in the size of the DAG.

Some additional terminology: for each pair (c_1, c_2) of conditional vertices in G_i, we refer to the subgraph of G_i beginning at c_1 and ending at c_2 as a *conditional construct* in G_i. Each conditional construct represents a branching choice in the code that is being modeled by G_i. A canonical conditional construct with branching factor 2 is depicted pictorially in Figure 1. It is permitted in our model that conditional

[1] Note that any DAG with multiple sources and/ or sinks can trivially be transformed to satisfy this requirement, by perhaps adding a dummy source and/ or a dummy sink. Hence adding this requirement does not reduce the expressiveness of the sporadic DAG tasks model.

constructs be nested: a conditional construct may contain additional conditional constructs within it. (Later, we will use the term *inner-most conditional construct* to denote a conditional construct that does not contain any conditional constructs within it.)

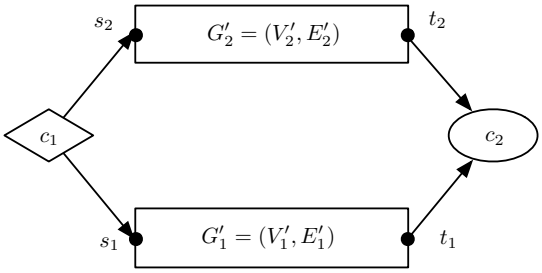


Fig. 1. A canonical conditional construct with branching factor 2. Vertices s_1 and t_1 (vertices s_2 and t_2, resp.) are the sole source vertex and sink vertex of G'_1 (G'_2, resp.).

The DAG for an example conditional sporadic DAG task is depicted in Figure 2. Small solid circles denote "dummy" vertices, which correspond to jobs with wcet equal to zero. Diamond and oval vertices denote start and end conditional vertices, respectively; there are two pairs of conditional vertices in this DAG, each with branching factor equal to two. (For those reading this on a color medium, the upper conditional construct is represented in blue; the lower conditional construct in red.) The semantics of this DAG task are as follows. Whenever a dag-job is released, the dummy source vertex has zero execution requirement and therefore immediately completes execution. Two vertices, with wcets 3 and 6 respectively, both become eligible for execution. Once both these jobs have completed, three jobs become eligible simultaneously.

- A conditional expression with a wcet of 1 is evaluated; depending upon the outcome of this evaluation, either three jobs each with wcet equal to 8, or two jobs each with wcet equal to 10, are executed.
  After these jobs complete, a single job with wcet equal to 12 is executed.
- Another conditional expression that has a wcet of 2 is evaluated. Depending upon the outcome of this evaluation, either a single job with a wcet equal to 8, or two jobs with wcet equal to 4 and 6 respectively, are executed.
- A single job with wcet equal to 12 is executed.

We seek to schedule a given collection of n conditional sporadic DAG tasks $\tau_1, \tau_2, \ldots, \tau_n$ upon m unit-speed processors. In this paper, we restrict our attention to *constrained deadline* systems: those in which $D_i \leq T_i$ for all tasks τ_i in the system.

Some additional notation and terminology:

- A *chain* in DAG task τ_i is a sequence of vertices $v_1, v_2, \ldots, v_k \in V_i$ such that (v_j, v_{j+1}) is an edge in G_i, $1 \leq j < k$. The *length* of this chain is defined to be the sum of the wcets of all the vertices in the chain. We denote by len_i the length of the longest chain in the DAG G_i. For our example DAG task τ_1 of Figure 2, $\mathit{len}_1 = (6 + 1 + 10 + 0 + 12) = 29$.
- Let $\mathcal{J}_i$ denote all possible complete collections of jobs that comprise a single dag-job of τ_i. Thus each $J \in \mathcal{J}_i$ denotes a collection of jobs obtained by completely executing through the DAG G_i once, taking into account the conditional branches within it. Observe that for each $J \in \mathcal{J}_i$, all the jobs in J have a common release time and a common deadline. Note, too, that $|\mathcal{J}_i|$ may be exponential in the number of vertices in G_i.
- Let vol_i be the maximum total wcet of a dag-job that could be generated by τ_i, taking into account the conditional branches within it. I.e., vol_i is the maximum, over all $J \in \mathcal{J}_i$, of the sum of the wcets of all the jobs in J. For our example DAG task τ_1 of Figure 2, a dag-job has maximum total wcet if the upper branch $(1 + (8 \times 3) + 0 = 25)$ is taken for the upper conditional, and the lower branch $(2+(4+6)+0 = 12)$ for the lower conditional. The total maximum wcet is therefore $6+3+25+12+12+12 = 70$, meaning that $\mathit{vol}_1 = 70$.
- For each task τ_i we define a *density* $\delta_i = \mathit{len}_i / D_i$ and a *utilization* $U_i = \mathit{vol}_i / T_i$.
- For a DAG sporadic task system τ we define its *maximum density* $\delta_{\max}(\tau)$ to be the largest density of any task in τ: $\delta_{\max}(\tau) = \max_{\tau_i \in \tau}\{\delta_i\}$; and its *utilization* $U(\tau)$ to be the sum of the utilizations of all the tasks in τ: $U(\tau) = \sum_{\tau_i \in \tau} U_i$.

For a traditional (non-conditional) sporadic DAG task system, the volume is equal to the sum of the wcet parameters of all the vertices in the DAG. Computing the volume of a conditional sporadic DAG task is not as simple as it was for traditional sporadic DAG tasks; in Section V-A, we will describe how volume may be determined efficiently for conditional sporadic DAG tasks. Once this is done, it is not difficult to see that (with the exception of $\mathcal{J}_i$), the remaining parameters defined above – the length, density, and utilization of each task, and the maximum density and total utilization of the task system – can be computed very efficiently, in time linear in the representation of the task system.

III. The GEDF scheduling of traditional sporadic DAG task systems

Some recent work [4], [8] has provided insight into the multiprocessor GEDF scheduling of collections of traditional (i.e., not conditional) sporadic DAG tasks. It was shown that GEDF has a speedup bound of $(2-1/m)$ when implemented upon an m-processor platform. In addition, a pseudo-polynomial time GEDF-schedulability test with speedup bound of $(2-1/m+\epsilon)$ for any constant $\epsilon > 0$ was derived in [4]. We now briefly review some techniques and results from [4]; we will subsequently extend these techniques and results to the GEDF schedulability analysis of systems of conditional sporadic DAG tasks.

The notion of the *work function* [4] serves to characterize the amount of work that could be generated by a sporadic

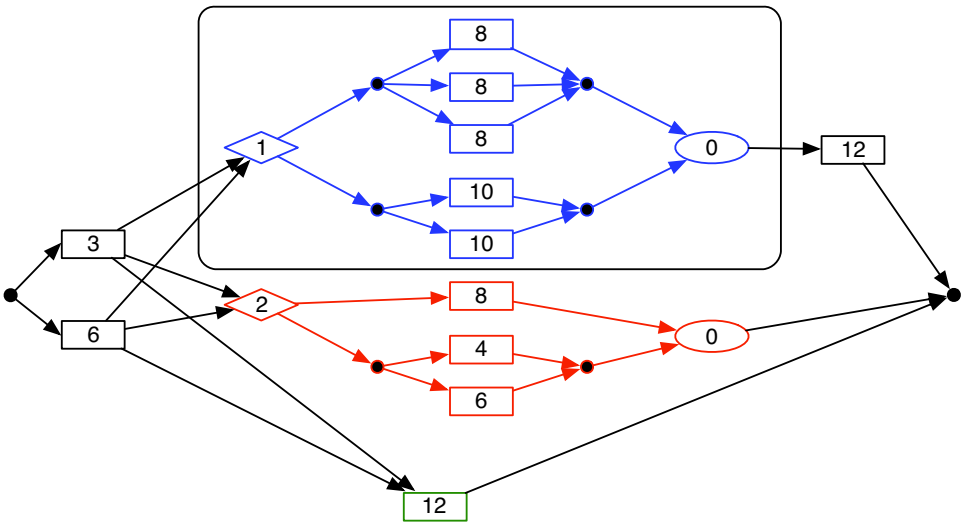


Fig. 2. The DAG of an example conditional sporadic DAG task. Vertices denote jobs; the numbers within vertices denote the wcets of the jobs. Small solid circles denote jobs with wcet equal to zero. Diamonds and ovals denote conditional start and end vertices respectively, and rectangles denote non-conditional vertices. (The large rectangle encloses a single conditional construct in the DAG that will be referenced later in this document.)

DAG task. Let s denote any positive real number. Suppose that we had an infinite number of speed-s processors available upon which to execute a given sporadic DAG task system τ. Let J denote any collection of jobs generated by τ. Let $S_\infty(J, s)$ denote the schedule obtained by allocating a speed-s processor to each job in J the instant it is ready to execute, and executing this job upon the allocated processor until it completes execution.

Definition 1 (**The** *work* **function).** *Let τ_i denote a sporadic DAG task, and s any positive real number ≤ 1. Let J denote any collection of jobs that is legally generated by τ_i.*

For any interval I, let work(J, I, s) *denote the amount of execution occurring within the interval I in the schedule $S_\infty(J, s)$, of jobs with deadlines that fall within I.*

For any positive integer t, let work(J, t, s) *denote the maximum value* work(J, I, s) *can take, over any interval I of duration equal to t.*

Finally, let work(τ_i, t, s) *denote the maximum value of* work(J, t, s)*, over all job sequences J that may legally be generated by the sporadic DAG task τ_i.*

That is, *work*(τ_i, t, s) is defined as the largest value, over all job sequences J that may be generated by τ_i, of the amount of execution occurring within some interval of duration t in the schedule $S_\infty(J, s)$, of jobs in J *that have deadlines within this interval*. Observe that schedule $S_\infty(J, s)$ executes each job as soon as it becomes available, thereby leaving as little work to be done later as possible. Hence any schedule for J on speed-s processors that meets all deadlines must complete at least *work*(J, I, s) units of execution over the interval I; this means that any schedule that can always meet all deadlines of τ_i must be able to execute τ_i's jobs for at least an amount *work*(τ_i, t, s) over an interval of size t.

The main result of [4] may be stated as follows.

Theorem 1 ([4]). *Sporadic DAG task system τ is GEDF schedulable on m unit-speed processors if there is a constant σ, $\delta_{\max}(\tau) \leq \sigma \leq 1$, such that*

$$\sum_{\tau_i \in \tau} \textit{work}(\tau_i, t, \sigma) \leq (m - (m-1)\sigma) \times t \qquad (1)$$

for all values of $t \geq 0$.

Hence to show that a given τ is EDF-schedulable upon m unit-speed processors it suffices, according to Theorem 1 above, to produce a value for σ such that Condition 1 holds for all $t \geq 0$. The schedulability test presented in [4] essentially reduces to determining whether $\sigma \leftarrow m/(2m-1)$ is such a value. It was shown in [4] that this fact establishes a speedup bound of $(2-1/m)$ for the global EDF scheduling of sporadic DAG task systems:

Corollary 1 ([4]). *GEDF has a speedup factor of $\left(2 - \frac{1}{m}\right)$ when scheduling systems of sporadic DAG tasks upon m preemptive processors.*

Schedulability testing in pseudo-polynomial time. As stated above, the schedulability test presented in [4] essentially consists of determining whether $\sigma \leftarrow m/(2m-1)$ causes Condition 1 to hold for all $t \geq 0$. As is evident from the statement of Theorem 1, executing this GEDF schedulability test requires the computation of the *work* function *work*$(\tau, t, m/(2m-1))$, for multiple values of t. It was shown in [4] how this could be done efficiently in pseudo-polynomial time at the cost

of an additional speedup ϵ, for any constant $\epsilon > 0$; the resulting pseudo-polynomial time GEDF-schedulability test has a speedup factor equal to $(2 - 1/m + \epsilon)$.

IV. GEDF scheduling of conditional sporadic DAG task systems: Properties

In this section and the next, we describe how the concepts and results described in Section III above may be extended to apply to the scheduling and schedulability analysis of *conditional* sporadic DAG task systems. In this section we extend the definition of the *work* function to conditional sporadic DAG tasks, and establish that Theorem 1 and Corollary 1 continue to hold for conditional sporadic DAG task systems. In Section V we will derive a pseudo-polynomial GEDF schedulability test with speedup factor $(2 - 1/m + \epsilon)$ for any constant $\epsilon > 0$, for conditional sporadic DAG task systems.

A work function for conditional sporadic DAG tasks. The second sentence of the definition of the work function (Definition 1) defines J as any collection of jobs that is legally generated by the sporadic DAG task τ_i. The entire Definition 1 extends without modification to conditional sporadic DAG tasks, with the understanding that this notion of "legal" collections of jobs must satisfy the semantics of conditional, rather than traditional, sporadic DAG tasks.

Extending Theorem 1 and Corollary 1. Inspection of the proofs of Theorem 1 and Corollary 1 in [4] reveal that both are based on considering worst-case behavior over all collections of jobs that could legally be generated by a given task system. And the only property of these collections of jobs that is needed in these proofs is that if there is a precedence constraint between a pair of jobs, then these jobs both have the same release date and the same deadline. This property is clearly satisfied by collections of jobs generated by systems of conditional sporadic DAG tasks; the proofs therefore go through unchanged. Hence, we are immediately able to conclude Theorem 2 and Corollary 2 as direct extensions of Theorem 1 and Corollary 1 respectively to the conditional sporadic DAG task model.

Theorem 2. *Conditional sporadic DAG task system τ is GEDF schedulable on m unit-speed processors if there is a constant σ, $\delta_{\max}(\tau) \leq \sigma \leq 1$, such that*

$$\sum_{\tau_i \in \tau} \textit{work}(\tau_i, t, \sigma) \leq (m - (m-1)\sigma) \times t$$

for all values of $t \geq 0$.

Corollary 2. *GEDF has a speedup factor of $\left(2 - \frac{1}{m}\right)$ when scheduling systems of conditional sporadic DAG tasks upon m preemptive processors.*

V. A GEDF schedulability test for conditional sporadic DAG task systems

Corollary 2 establishes that from the perspective of processor speedup factor, generalizing the sporadic DAG task model to allow for the additional representation of conditional constructs incurs no additional cost or penalty – this additional capability is, in essence, had "for free." However, this result does not in itself tell us how we may *test* a given conditional sporadic DAG task system for GEDF-schedulability; in this section we will derive such an efficient (pseudo-polynomial time) GEDF schedulability test for conditional sporadic DAG task systems.

We find it helpful to first define an additional function for constrained-deadline sporadic DAG tasks called the *remaining demand function*, denoted *rdem*. This is defined as follows. Recall that (i) $\mathcal{J}_i$ denotes all possible complete collections of jobs that comprise a single dag-job of τ_i; (ii) for each $J \in \mathcal{J}_i$, all the jobs in J have have a common release date and a common deadline – the release date and deadline of the dag-job that generates them; and (iii) $S_\infty(J, s)$ denotes, for any collection of jobs J, a schedule obtained by allocating a speed-s processor to each job in J the instant it is ready to execute, and executing this job upon its allocated processor until it completes execution.

Definition 2 (The *rdem* function $\textit{rdem}(\tau_i, t, s)$). *Consider any $J \in \mathcal{J}_i$ for a given conditional sporadic DAG task τ_i, and let t denote any positive real number $\leq D_i$. Let* rdem(J, t, s) *denote the amount of work remaining to be executed — i.e., the sum of the* wcet*'s of all the jobs in J* minus *the amount of execution that has already occurred — in schedule $S_\infty(J, s)$ a duration t time units after the (common) release date of the jobs in J. $\textit{rdem}(\tau_i, t, s)$ is defined to be the maximum value of* rdem(J, t, s) *over all collections of jobs $J \in \mathcal{J}_i$.*

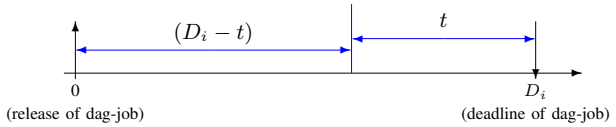
That is, $\textit{rdem}(\tau_i, t, s)$ denotes the maximum amount of work that could remain to be executed, if a single dag-job of task τ_i were to execute for t time units upon infinitely many speed-s processors.

The significance of the *rdem* function arises from its relationship with the *work* function, as identified in the following lemma.

Lemma 1. *Let τ_i denote a constrained-deadline conditional sporadic DAG task. For any $s \geq \delta_i$ and for all $t \leq D_i$*

$$\textit{work}(\tau_i, t, s) = \textit{rdem}(\tau_i, D_i - t, s) \tag{2}$$

Proof: Recall (Definition 1) that $\textit{work}(\tau_i, t, s)$ is defined as the *maximum* value, over all job sequences J that may be generated by τ_i, of the amount of execution occurring within some interval of duration t in the schedule $S_\infty(J, s)$, of jobs in J that have deadlines within this interval. For intervals of duration $\leq D_i$, it is evident that this maximum is achieved when some dag-job of τ_i has a deadline that coincides with the rightmost endpoint of the interval of duration t:



Upon infinitely many speed-s processors, the maximum work

remaining to be done $(D_i - t)$ time units after the dag-job's arrival is, by definition, $\mathit{rdem}(\tau_i, D_i - t, s)$. It is evident from the definition of the *work* function and from the picture above that this is also equal to $\mathit{work}(\tau_i, t, s)$. □

Let us take a closer look at the *work* function $\mathit{work}(\tau_i, t, s)$ for a conditional sporadic DAG task τ_i and $s \geq \delta_i$. It had been pointed out in [4] that the scenario defining the value of $\mathit{work}(\tau_i, t, s)$ has the deadline of some dag-job of τ_i coincide with the rightmost endpoint of an interval of duration t, and the other dag-jobs of τ_i released as closely as possible. There will be $\lfloor t/T_i \rfloor$ complete dag-jobs of τ_i within such an interval (this is illustrated in Figure 3 for an example task with $D_i = 15$ and $T_i = 20$, for $t \leftarrow 70$). The maximum amount of *work* for each such dag-job is clearly equal to vol_i (recall that vol_i denotes the maximum total wcet of any dag-job that could be generated by τ_i, taking into account the conditional branches within it). It remains to determine the amount of execution occurring in the remaining part of the interval (which is of duration $t \bmod T_i$) on jobs that have deadlines within this part of the interval (see again Figure 3 – the interval left over after the $\lfloor t/T_i \rfloor = \lfloor 70/20 \rfloor = 3$ complete dag-jobs have been accounted for is the interval [5,15], of duration $(70 \bmod 20) = 10$).

Now if $(t \bmod T_i)$ is $\geq D_i$, then it is evident that an entire dag-job of τ_i will be accommodated within this interval, thereby contributing an amount vol_i to $\mathit{work}(\tau_i, t, s)$. We therefore have the following relationship:

$$\mathit{work}(\tau_i, t, s) = \mathit{vol}_i \times \lfloor t/T_i \rfloor + \begin{cases} \mathit{vol}_i, & \text{if } (t \bmod T_i) \geq D_i \\ \mathit{work}(\tau_i, t \bmod T_i, s), & \text{if } (t \bmod T_i) \leq D_i \end{cases}$$

Since Lemma 1 is applicable in the case where $(t \bmod T_i) \leq D_i$, we can replace $\mathit{work}(\tau_i, t \bmod T_i, s)$ in this case by $\mathit{rdem}(\tau_i, D_i - (t \bmod T_i), s)$, to finally obtain

$$\mathit{work}(\tau_i, t, s) = \mathit{vol}_i \times \lfloor t/T_i \rfloor + \begin{cases} \mathit{vol}_i, & \text{if } (t \bmod T_i) \geq D_i \\ \mathit{rdem}(\tau_i, D_i - (t \bmod T_i), s), & \text{if } (t \bmod T_i) \leq D_i \end{cases} \quad (3)$$

We have thus reduced the problem of computing $\mathit{work}(\tau_i, t, s)$ for all t to that of computing $\mathit{rdem}(\tau_i, t, s)$ for values of $t \leq D_i$. It remains to specify how $\mathit{rdem}(\tau_i, t, s)$ is to be computed for values of $t \leq D_i$. We will first illustrate this with an example, computing $\mathit{rdem}(\tau_i, t, 1)$ for an example task τ_i executing upon a platform of unit-speed processors. This example task has parameters $D_i = 15$ and $T_i = 20$, and a DAG G_i that is depicted in Figure 4. (Observe *en passant* that this DAG is the same as the part of the DAG in Figure 2 enclosed in a large rectangle, representing the upper conditional construct of that task.) According to G_i, a conditional expression having wcet=1 is evaluated each time a dag-job of τ_i is released. Depending upon the outcome of this evaluation, either three jobs of wcet 8 each that may execute in parallel, or two jobs of wcet 10 each that may execute in parallel, are to be executed. There is no execution cost (and hence no wcet) associated with recombining the branches; hence, the conditional vertex depicting the end of the conditional construct has a wcet of zero.

Let us determine $\mathit{rdem}(\tau_i, t, 1)$ as a function of t for this example task τ_i. $|\mathcal{J}_i| = 2$ for this task; i.e., there are two possible flows of control through this DAG for a single dag-job of this task, depending upon whether the upper or the lower branch is taken upon evaluation of the conditional expression. We separately consider the cases when the upper or the lower branches are taken; the resulting functions are depicted graphically in Figure 5.

- **The upper branch is taken.** The amount of work remaining is depicted as the line beginning at the point $(0, 25)$ in Figure 5 (the blue line, for those reading this on a color medium). At the beginning, there are 25 units of work remaining to be done. Only one job – the one corresponding to the conditional vertex – executes over the interval $[0, 1)$; hence the slope of the line during this interval is -1. Once the conditional expression has been evaluated, three jobs execute in parallel for eight time units; hence the slope is -3.
- **The lower branch is taken.** The amount of work remaining is depicted as the line beginning at the point $(0, 21)$ in Figure 5 (the red line). At the beginning, there are 21 units of work remaining to be done. As in the case above, only the job corresponding to the conditional vertex executes over the interval $[0, 1)$; hence the slope of the line during this interval is -1. Once the conditional expression has been evaluated, two jobs execute in parallel for ten time units; hence the slope is -2.

It is immediately evident from Figure 5 that for values of $t \leq 5$, executing the upper branch leaves more work remaining to be done after t time units (this is depicted as the blue line); for values of $t \geq 5$, executing the lower branch leaves more work remaining to be done (the red line). I.e., the upper envelope

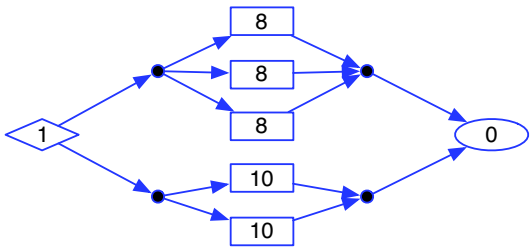


Fig. 4. The DAG G_i of an example conditional DAG task τ_i. For this task, $D_i = 15$ and $T_i = 20$. Note that $\mathit{len}_i = 11$ and corresponds to taking the lower branch of the conditional, while $\mathit{vol}_i = 25$ and corresponds to taking the upper branch.

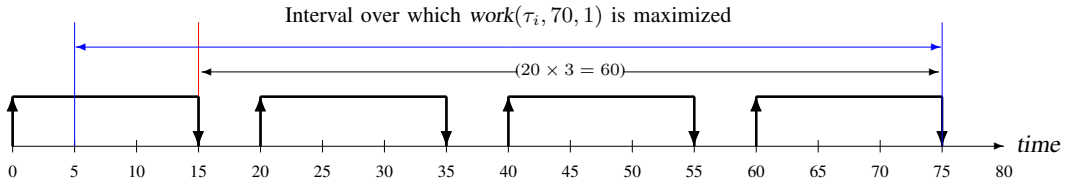


Fig. 3. Computing $work(\tau_i, 70, 1)$ for the task τ_i of Figure 4.

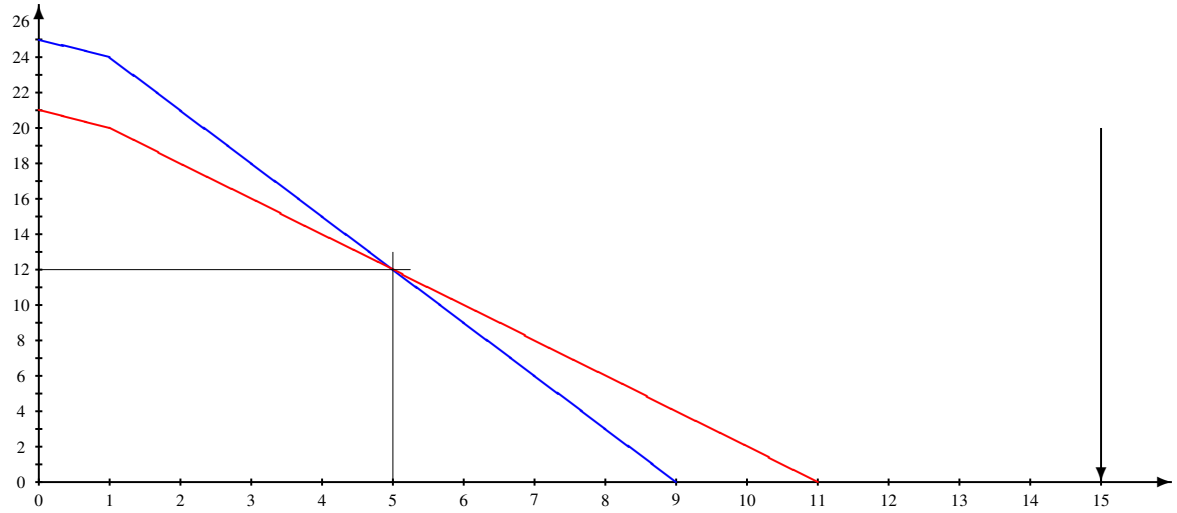


Fig. 5. Remaining work (y axis) as a function of time elapsed (x axis) since the release of a dag-job of the task τ_i.

of the two individual *rdem* functions[2] corresponding to the two different paths through the conditional code represents the maximum amount of remaining work for all values of t, and $rdem(\tau_i, t, 1)$ is therefore the upper envelope of the two individual *rdem* lines plotted in Figure 5.

(It is interesting to understand what this means: different paths through the conditional code represent the "worst case," in the sense of leaving the maximum amount of work remaining to be done, for different values of t. Hence it is not possible to identify one particular path through the code such that simply evaluating this path suffices for determining the worst-case behavior of the task for all values of t.)

Let us now apply Equation 3 to compute $work(\tau_i, t, 1)$ for the example task of Figure 4, for values of $t = 65, 70, 72,$ and 78. Observe that $\lfloor t/T_i \rfloor = 3$ for all four values of t, and so $vol_i \times \lfloor t/T_i \rfloor = 25 \times 3 = 75$ for all values of t. While $(t \bmod T_i)$ is $\leq D_i$ for $t = 65$, $t = 70$, and $t = 72$, it is $> D_i$ for $t = 78$. By separately applying Equation 3 for each value of t, we therefore get

$$\begin{aligned} work(\tau_i, 65, 1) &= 75 + rdem(\tau_i, 10, 1) = 75 + 2 = 77 \\ work(\tau_i, 70, 1) &= 75 + rdem(\tau_i, 5, 1) = 75 + 12 = 87 \\ work(\tau_i, 72, 1) &= 75 + rdem(\tau_i, 3, 1) = 75 + 18 = 93 \\ \text{and} \quad work(\tau_i, 78, 1) &= 75 + vol_i = 75 + 25 = 100 \end{aligned}$$

[2]The *upper envelope* of a collection of functions is defined to be the pointwise maximum of these functions.

The approach that we applied in computing the *rdem* function depicted in Figure 5 for our example above – compute it separately for both the possible flows of control through the DAG, and take the upper envelope of the two computed functions – is easily generalized to yield a simple methodology for computing the *rdem* function (and thereby the work function) for any constrained-deadline sporadic DAG task τ_i:

- Compute $rdem(J, t, s)$ for each $J \in \mathcal{J}_i$. It should be evident that this can be easily done in polynomial time for each given J; we omit the details.
- $rdem(\tau_i, t, s)$ is then simply the upper envelope of the individual $rdem(J, t, s)$ functions computed above.

Although this approach is correct, it suffers from the same problem as the multi-DAG model of Fonseca et al. [6]: $|\mathcal{J}_i|$ — the number of distinct possible flows of control, and hence the number of distinct collections of jobs J for which we would need to compute $rdem(J, t, s)$ — may be exponential in the size of the DAG; the overall algorithm would therefore take exponential time. We seek to do better; hence instead of explicitly computing the *rdem* function in this manner, we will exploit the insights we have gained above regarding the properties of the *rdem* function to develop a more efficient approach to GEDF schedulability analysis of conditional sporadic DAG task systems. Rather than seeking to directly determine whether a given task system satisfies Theorem 2 (doing so requires the explicit computation of the *work* function), we will instead efficiently *transform* each conditional sporadic DAG task τ_i to a non-conditional one τ_i'.

These two tasks will be "equivalent" in the sense that they will both have the same *len*, *vol*, deadline and period parameters, and they will satisfy the property that for all $t \geq 0$ and all $s \geq \delta_i$,

$$\mathit{work}(\tau_i, t, s) = \mathit{work}(\tau'_i, t, s) \tag{4}$$

It will then follow that a conditional sporadic DAG task system τ will satisfy Theorem 2 if and only if the non-conditional sporadic DAG task system τ' that is obtained by so transforming each task in τ satisfies Theorem 1. We can then apply the pseudo-polynomial time GEDF schedulability test for non-conditional sporadic DAG task systems of [4] to the non-conditional sporadic DAG task system τ', to obtain a pseudo-polynomial time GEDF schedulability test for conditional sporadic DAG task systems that has speedup factor $(2 - 1/m + \epsilon)$ for any constant $\epsilon > 0$.

We first illustrate this transformation for our example task of Figure 4. Consider a task τ_j with $D_j = D_i = 15$ and $T_j = T_i = 20$, that has the DAG G_j depicted in Figure 6. It is readily verified that the remaining work function of this task is identical to the upper envelope of the two remaining work functions depicted in Figure 5. Hence *tasks τ_i and τ_j have identical rdem functions* (and therefore, identical *work* functions), and τ_j is thus a non-conditional sporadic DAG task that is "equivalent" to τ_i in the sense that both have the same *work* functions.

How did we obtain τ_j? Essentially, by inspection of Figure 5; we set out to construct a non-conditional sporadic DAG task with *rdem* function equal to the upper envelope of the two *rdem* functions plotted in Figure 5:

- Since the upper envelope has a slope of -1 over the interval $[0, 1)$, we introduced a single vertex with wcet $= (1 - 0) = 1$.
- The slope of the upper envelope is then -3 over the interval $[1, 5)$; this is modeled by adding a second "layer" of three vertices, each with wcet $= (5 - 1) = 4$, as successor vertices.
- The slope of the upper envelope is subsequently -2 over the interval $[5, 11)$; this is modeled by adding a third layer of two vertices, each with wcet $= (11 - 5) = 6$, as successor vertices.
- The final layer with a single vertex with wcet $= 0$ represents the end of the conditional construct.
- (Notice that we have added edges from each vertex in each layer to all vertices in the immediately succeeding layer.)

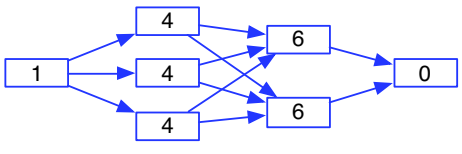


Fig. 6. The DAG G_j of an example conditional DAG task τ_j.

Before we comment further, observe two simple properties of the basic transformation defined above:

(**VP**) The volume of τ'_i is equal to the volume of τ_i.
(**LP**) The length of τ'_i is equal to the length of τ_i.

At first glance, this approach does not appear to offer any additional efficiency over the earlier methods, since in order to compute the upper envelope of the individual *rdem* function we must first explicitly compute these individual *rdem* functions for all the (possibly exponentially many) possible control flows. One further result is necessary in order to be able to achieve our goal of efficiently transforming a conditional sporadic DAG task to an equivalent non-conditional one; we illustrate the use of this result via an example, prior to formally proving it.

As we had pointed out earlier, it may be verified that the DAG in Figure 4 appears as one conditional construct in the larger DAG of Figure 2 – the one that is enclosed within a larger rectangle. If we were to replace that entire conditional construct in the DAG of Figure 2 with the DAG of Figure 6, we would obtain a conditional DAG with *one fewer conditional construct*, for which (by Theorem 3 below) the *work* function is identical to the *work* function of the DAG of Figure 2.

We can do likewise for the other (lower) conditional construct in the DAG of Figure 2; Figure 7 depicts the application of a similar transformation to this lower conditional construct. Finally, Figure 8 depicts the non-conditional DAG resulting from applying both transformations (and some cosmetic changes – deletions of the dummy source and sink vertices).

The transformation algorithm. We now describe our algorithm for transforming a conditional sporadic DAG task τ_i to an equivalent non-conditional sporadic DAG task τ'_i. $D'_i \leftarrow D_i$, and $T'_i \leftarrow T_i$. To obtain G'_i, we start out with the DAG G_i and repeatedly

1) identify an innermost conditional construct;
2) construct a non-conditional DAG that is equivalent to this innermost conditional construct (we describe below how this may be done); and
3) replace the identified innermost conditional construct with the constructed equivalent non-conditional DAG

until there are no remaining conditional constructs in the DAG.

Constructing an equivalent non-conditional DAG. Let us suppose that we seek a non-conditional DAG that is equivalent to an innermost conditional construct that is notated as in Figure 1.

- Separately construct the *rdem* functions for the collections of jobs corresponding to all the vertices in $\{c_1, c_2\} \cup V'_1$ and $\{c_1, c_2\} \cup V'_2$ respectively. Each *rdem* is piecewise linear, with the number of linear segments bounded from above by the number of vertices in the graph, and each linear segment has a negative integer slope.
- Determine the upper envelope of these two *rdem* functions. This upper envelope is piecewise linear, each linear

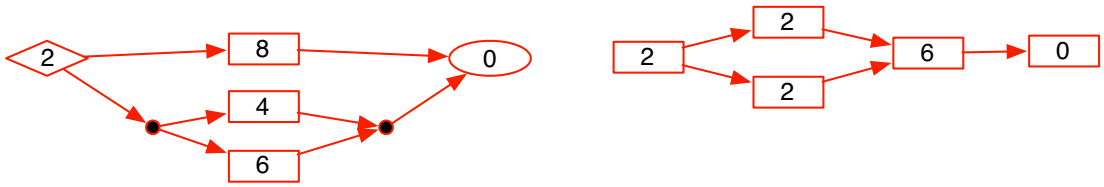


Fig. 7. Transforming the lower conditional construct of the DAG of Figure 2.

segment has a negative integer slope, and the total number of linear segments is bounded from above by the number of vertices in $\{c_1, c_2\} \cup V_1' \cup V_2'$.

- Construct a DAG $G'' = (V'', E'')$ that has the same *rdem* function as the upper envelope determined above. This graph is constructed as a "layered" one, with as many layers as there are linear segments in the upper envelope plus 1. The number of vertices in the k'th layer is equal to the (negation of the) slope of the k'th segment of the upper envelope, and each vertex is labeled with a wcet equal to the duration of the time axis spanned by this k'th segment. The last layer consists of a single sink vertex with wcet $= 0$. There is an edge from each vertex to each vertex in the immediately succeeding layer.

This DAG $G'' = (V'', E'')$ is the equivalent non-conditional DAG.

Theorem 3. *Let τ_i denote a constrained-deadline conditional sporadic DAG task, and $\hat{\tau}_i$ denote the (perhaps conditional) sporadic DAG task obtained by replacing an innermost conditional construct in the DAG G_i of τ_i by an equivalent non-conditional DAG as described above.*
For all $t, 0 \leq t \leq D_i$, rdem(τ_i, t, s) = rdem$(\hat{\tau}_i, t, s)$ (and therefore work(τ_i, t, s) = work$(\hat{\tau}_i, t, s)$ for all t as well).

Proof Sketch: Let us assume that the innermost conditional construct that is replaced in τ_i is notated as in Figure 1. Recall that $\mathcal{J}_i$ denotes all possible complete collections of jobs that comprise a single dag-job of τ_i. Consider any pair $J_1 \in \mathcal{J}_i, J_2 \in \mathcal{J}_i$ of such complete collections of jobs comprising a single dag-job of τ_i, that differ only in the choices they make with regard to the conditional construct that is selected for replacement. That is, exactly one of J_1, J_2 contains (jobs corresponding to) all the vertices in V_1', while the other contains (jobs corresponding to) all the vertices in V_2'; other than these differences, they both contain (jobs corresponding to) exactly the same collection of vertices.

Let us denote the difference between the sum of the wcets of all the jobs in V_1' and the sum of the wcets of all the jobs in V_2' by Δ.

Consider the functions *rdem*(J_1, t, s) and *rdem*(J_2, t, s) as functions of t. At time-instant 0, these differ by exactly an amount equal to Δ. Observe that the schedules $S_\infty(J_1, s)$ and $S_\infty(J_2, s)$ are identical prior to the instant that they both begin the execution of the job corresponding to the conditional vertex c_1 of Figure 1; hence at that instant (let us denote this instant as t_o), we have that *rdem*(J_1, t_o, s) and *rdem*(J_2, t_o, s) differ by exactly Δ.

Examining the schedules $S_\infty(J_1, s)$ and $S_\infty(J_2, s)$ at times $> t_o$ and prior to the execution of vertex c_2 in either schedule, we observe that

- Those jobs that belong in both J_1 and J_2 execute at the same times in both schedules; hence, the decrease in the remaining demand due to the execution of these jobs proceeds in exactly the same manner in both *rdem*(J_1, t, s) and *rdem*(J_2, t, s).
- Of course, the execution of the jobs belonging to exactly one of J_1 or J_2 proceeds differently in the schedules $S_\infty(J_1, s)$ and $S_\infty(J_2, s)$; the manner in which these executions happen is represented in the *rdem* functions that were separately constructed for the collections of jobs corresponding to the vertices in $\{c_1, c_2\} \cup V_1'$ and $\{c_1, c_2\} \cup V_2'$ respectively.

At times $> t_o$ and prior to the execution of vertex c_2 in either schedule, we can therefore consider the *rdem* functions for each of J_1 and J_2 as the sum of a part that is identical in both, and a part that is equal to the *rdem* functions that were separately constructed for the collections of jobs corresponding to the vertices in $\{c_1, c_2\} \cup V_1'$ and $\{c_1, c_2\} \cup V_2'$ respectively. And as was argued in the case when we considered a single conditional construct in isolation (and therefore had only two possible flows of control), the upper envelope of both individual *rdem* functions represents a tight upper bound on the *rdem* function over both the flows of control that are represented by the part of J_1 and J_2 that differ from each other. The correctness of the theorem follows from the observation that by construction, the DAG $G'' = (V'', E'')$ that replaces the conditional construct has an *rdem* function exactly equal to this upper envelope.

□

A. Computing vol$_i$ for conditional sporadic DAG tasks

We had stated in Section II that while the volume of a traditional sporadic DAG task is simply equal to the sum of the wcet parameters of all the vertices in the DAG, determining the volume of a conditinoal sporadic DAG task is not quite as straightforward. The approach we adopt to computing it is to first transform the conditional sporadic DAG task to a non-conditional sporadic DAG task, as described in Section V above. (Note that this transformation is needed in any case

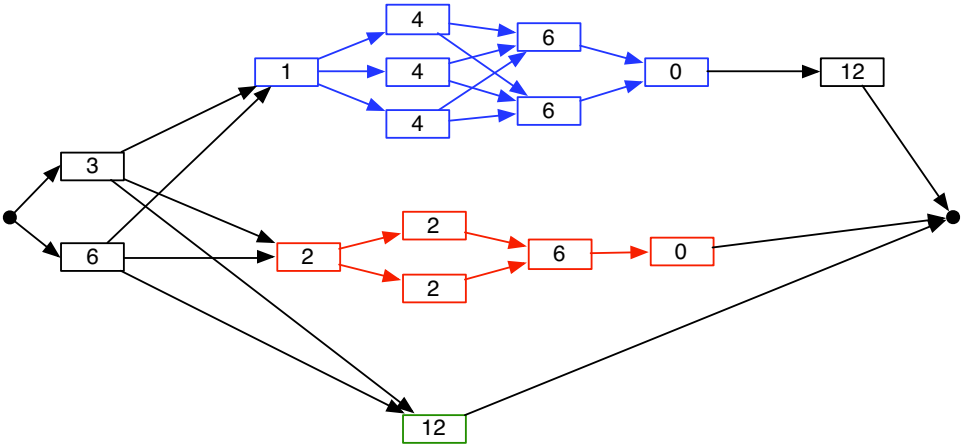


Fig. 8. The DAG of Figure 2 with both conditional constructs removed.

for GEDF schedulability analysis.) Once the transformation has been completed, we obtain the volume of the conditional sporadic DAG task by simply adding the wcets of all the vertices in the transformed, non-conditional, DAG.

VI. Summary and Conclusions

We have proposed and evaluated a task model for representing recurrent real-time task systems for execution upon multiprocessor platforms, that is capable of both (i) exposing internal parallelism in the task workload to the scheduling mechanism; and (ii) accurately modeling conditional constructs that may be present within individual tasks. This model, the conditional sporadic DAG tasks model, is a strict generalization of the sporadic DAG tasks model [3]. We show that GEDF is a suitable algorithm for the run-time scheduling of conditional sporadic DAG task systems — it has a speedup factor of $(2 - 1/m)$ upon m-processor platforms, and we present a sufficient GEDF-schedulability test that has pseudo-polynomial run-time and a speedup factor $(2 - 1/m + \epsilon)$ for any constant $\epsilon > 0$.

Acknowledgements

Supported in part by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and a grant from General Motors Corp.

References

[1] B. Andersson and D. de Niz. Analyzing global-edf for multiprocessor scheduling of parallel tasks. In *Principles of Distributed Systems, 16th International Conference, OPODIS 2012, Rome, Italy, December 18-20, 2012. Proceedings*, pages 16–30, 2012.

[2] S. Baruah. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014)*, Madrid, Spain, July 8-11, 2014.

[3] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS 2012, pages 63–72, San Juan, Puerto Rico, 2012.

[4] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG task model. In *Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems*, ECRTS '13, pages 225–233, Paris (France), 2013.

[5] H. S. Chwa, J. Lee, K.-M. Phan, A. Easwaran, and I. Shin. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *Proceedings of the 25nd Euromicro Conference on Real-Time Systems (ECRTS 2013)*, Paris, France, July 9-12, 2013.

[6] J. Fonseca, V. Nelis, G. Raravi, and L. M. Pinho. A Multi-DAG model for real-time parallel applications with conditional execution. In *Proceedings of the ACM/ SIGAPP Symposium on Applied Computing (SAC)*, Salamanca, Spain, April 2015. ACM Press.

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

[8] J. Li, K. Agrawal, C. Lu, and C. D. Gill. Analysis of global EDF for parallel tasks. In *Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems*, ECRTS '13, pages 3–13, Paris (France), 2013.

[9] J. Li, J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 85–96, 2014.

[10] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11-13, 2012*, pages 321–330, 2012.

[11] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet. Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS 2013)*, Sophia Antipolis, France, October 16-18, 2013.

[12] A. Saifullah, K. Agrawal, C. Lu, and C. D. Gill. Multi-core real-time scheduling for generalized parallel task models. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*, pages 217–226, 2011.

[13] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill. Parallel real-time scheduling of dags. *IEEE Trans. Parallel Distrib. Syst.*, 25(12):3242–3252, 2014.