The federated scheduling of constrained-deadline sporadic DAG task systems

Sanjoy Baruah
The University of North Carolina at Chapel Hill
baruah@cs.unc.edu

***Abstract*—In the *federated* approach to multiprocessor scheduling, a task is either restricted to execute upon a single processor (as in partitioned scheduling), or has exclusive access to any processor upon which it may execute. Earlier studies concerning the federated scheduling of task systems represented using the sporadic DAG model were restricted to implicit-deadline task systems; the research reported here extends this study to the consideration of task systems represented using the more general *constrained*-deadline sporadic DAG model.**

I. Introduction

Many real-time systems are modeled as a finite collection of independent recurrent tasks, each of which generates a potentially infinite sequence of jobs. Each job is characterized by a release or arrival time, a worst-case execution time (WCET), and a deadline, with the interpretation that the job needs to receive an amount of execution that may be as large as its WCET, in the interval between its arrival time and its deadline. The different tasks in a task system are assumed to be independent of each other; hence, jobs of different tasks may execute simultaneously on different processors upon a multiprocessor platform. Since its origins in the late 1960's and early 1970's, real-time scheduling theory has primarily been concerned with determining how multiple such recurrent tasks can be scheduled on a shared platform. Many models, representing different choices in the trade-off between expressiveness of the model and the complexity of performing analysis of systems expressed using the model, have been considered in the literature, including the simple *Liu and Layland* [19] model, the *three-parameter sporadic* [21] model, the *multiframe* [22] model, the *generalized multiframe* [6] model, the *recurrent real-time task* [3] model, the *digraph* [25] model, etc. — see, e.g., [24] for an excellent current survey on models for representing recurrent real-time tasks.

The sporadic DAG task model. All the task models listed above share the property that parallelism within an individual task is forbidden: each task is assumed to represent a single thread of computation, which may execute upon at most one processor at any point in time. This has traditionally not been perceived as a shortcoming of these models, since these models were introduced in the context of uniprocessor systems in which parallel execution is not possible in any case. More recently, the trend towards implementing real-time systems upon multiprocessor and multicore platforms has given rise to a need for models capable of exposing parallelism that may exist within the workload to the scheduling mechanism. There has therefore been a move towards designing new models that allow for the expression of such partial parallelism within individual tasks, thereby permitting multiple jobs of a single task to be executed simultaneously upon multiple processors. Earlier models of this form include the moldable [20], the malleable [10], and the parallel synchronous [14] tasks models. The sporadic DAG tasks model [5] is a generalization of these earlier models, and has come to be the focus of much study within the real-time scheduling community over the last couple of years. In brief (we provide a detailed description in Section II) each task in this model is modeled as a directed acyclic graph (DAG) $G = (V, E)$. Each vertex of the DAG corresponds to a sequential job, and is characterized by a worst-case execution time (WCET). Each (directed) edge of the DAG represents a precedence constraint: if (v, w) is an 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 may execute in parallel. 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, all $|V|$ of the jobs become available for execution simultaneously, subject to the precedence constraints. During any given execution 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.

One significant benefit of the sporadic DAG tasks model over earlier sequential models is that it can be thought of as representing a *higher level abstraction* than the sequential models — complex multi-threaded computations that are naturally expressed as directed acyclic graphs may be represented as a single task using this model, without the need, as was the case with the earlier sequential models, to be artificially broken up into smaller sequential tasks.

Federated scheduling. Real-time systems have traditionally been implemented upon multiprocessor platforms using either the partitioned or the global approach. In the *partitioned* approach, each task gets mapped on to a processor prior to

978-3-9815370-4-8/DATE15/©2015 EDAA

run-time; during run-time, all jobs generated by a task may execute only upon the processor to which the task has been mapped. In the *global* approach, by contrast, different jobs of the same task may execute upon different processors, and if preemption is permitted then a preempted job may resume execution upon a processor different from the one on which it had previously been executing. It has been observed that as a general rule, partitioned scheduling has the advantage of simplicity of analysis and implementation, whereas global scheduling allows for the use of a larger fraction of the total platform computing capacity.

Although the notions of partitioned and global scheduling pre-date the introduction of task models capable of representing multi-threaded tasks, it is clear that the global approach is also applicable to task systems represented using the sporadic DAG task model and the other models that permit intra-task parallelism; a number of papers (e.g., [23], [16], [5], [8], [1]) have studied the global scheduling of such task systems. However, it is not immediately evident what the natural analog of partitioned scheduling should be for systems represented using these models. If we were to restrict all jobs of a task to execute upon only a single processor, we would hobble the expressiveness of the model considerably by forbidding tasks with a (parallelizable) computational demand exceeding the capacity of a single processor to accommodate it. The federated scheduling [17] approach appears a reasonable extension of partitioned scheduling to sporadic DAG task systems; in this approach *tasks that are permitted to execute upon more than one processor are granted exclusive access to the processors upon which they execute*, while the remaining tasks are partitioned amongst a pool of shared processors[1]. The simplicity of partitioned scheduling is thus maintained, while we will see that for some task models the capacity loss due to partitioning is capped at no larger a fraction of the platform capacity than was the case with partitioned scheduling of sequential tasks.

Organization. The remainder of this paper is organized as follows. In Section II we formally define the sporadic DAG tasks model and list the contributions of this paper. In Section III we summarize the state of knowledge concerning the federated scheduling of sporadic DAG task systems resulting as a consequence of this paper and prior research. We describe our algorithm for federated scheduling in Section IV.

II. The sporadic DAG tasks model

We consider the federated scheduling of sporadic DAG task systems upon a preemptive identical multiprocessor platform. A task system τ in the sporadic DAG tasks model is comprised of a number of independent sporadic DAG tasks: $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i 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 real numbers. We discuss these parameters below.

- 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 (it is required that these edges do not form any cycle). Each $v \in V_i$ denotes a sequential operation (a "job"). Each job $v \in V_i$ is characterized by a worst-case execution time (WCET) $e_v \in \mathbf{N}$. 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. (We say a job becomes *available* — i.e., eligible to execute — once all its predecessor jobs have completed execution.)
- A *release* or arrival of a *dag-job* of the task at time-instant t means that all $|V_i|$ jobs $v \in V_i$ are released at time-instant t. The period T_i denotes the minimum amount of time that must elapse between the release of successive dag-jobs.
- If a dag-job is released at time-instant t then all $|V_i|$ jobs that were released at time-instant t must complete execution by time-instant $t + D_i$.

Analogously with 3-parameter sporadic tasks, we may define a system τ of sporadic DAG tasks to be *implicit-deadline* if $D_i = T_i$ for all $\tau_i \in \tau$, *constrained-deadline* if $D_i \leq T_i$ for all $\tau_i \in \tau$, and *arbitrary-deadline* otherwise.

Some additional notation and terminology:

- A *chain* in the sporadic DAG task τ_i is a sequence of nodes $v_1, v_2, \ldots, , v_k$ in G_i such that (v_i, v_{i+1}) is an edge in G_i, $1 \leq i < k$. The length of this chain is defined to be the sum of the WCETs of all its nodes: $\sum_{i=1}^{k} e_{v_k}$.
- len_i denotes the length of the longest chain in G_i. We point out that len_i can be computed in time linear in the number of vertices and the number of edges in G_i, by first obtaining a topological sorting of the vertices of the graph and then running a straightforward dynamic program.
- $\mathrm{vol}_i = \sum_{v \in V_i} e_v$ denotes the total WCET of each dag-job of the task τ_i. Note that vol_i can be computed in time linear in the number of vertices in G_i.
- The *utilization* u_i of task τ_i is defined as $u_i \stackrel{\text{def}}{=} \mathrm{vol}_i / T_i$. A task with utilization ≥ 1 is called a *high-utilization task*; one with utilization < 1 is called a *low-utilization task.* (This terminology is from [17].)
- The *density* δ_i of task τ_i is defined as $\delta_i \stackrel{\text{def}}{=} \mathrm{vol}_i / \min(D_i, T_i)$ Analogously to the above, a task with density ≥ 1 is called a *high-density task*; one with density < 1 is called a *low-density task.*
- We define the utilization $U_{\mathrm{sum}}(\tau)$ of task system τ to be the sum of the utilizations of all the tasks in τ: $U_{\mathrm{sum}}(\tau) = \sum_{\tau_i \in \tau} u_i$.

Example 1: An example sporadic DAG task τ_1 is depicted graphically in Figure 1. The DAG G_i for this task consists

[1]This differs somewhat from the definition of federated scheduling introduced in [17]: there, tasks with computational demand exceeding the capacity of a single processor are granted exclusive access to several processors, but it remains unspecified how tasks with computational demand no larger than the capacity of a single processor should be scheduled. Since in this paper we are looking upon federated scheduling as a generalization of partitioned scheduling to sporadic DAG task systems, it makes sense to restrict the scheduling of the tasks that may share processors to partitioned scheduling only.

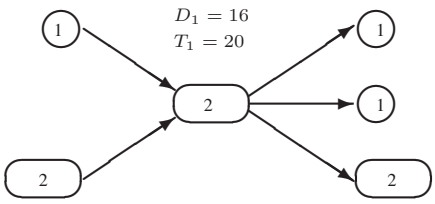

Fig. 1. Example sporadic DAG task τ_1. Vertices are labeled with the WCET's of the jobs they represent (the WCET is also indicated by the size of the vertex).

of five vertices and five directed edges denoting precedence constraints. The longest chain is of length $\text{len}_1 = 6$; By summing all the WCET's, we can see that $\text{vol}_1 = 9$. Also $\delta_1 = 9/16$, and $u_1 = 9/20$; since $\delta_1 < 1$, task τ_1 is a low-density task.

Performance measures. *Speedup bounds* have come to be widely used as a metric for quantifying the effectiveness of multiprocessor scheduling algorithms. In the context of the federated scheduling of sporadic DAG systems, they may defined as follows:

Definition 1 (Speedup bound): An algorithm A for the federated scheduling of sporadic DAG task systems is said to have a *speedup bound* b if any task system that can be scheduled on m speed-1 processors by an optimal clairvoyant federated scheduling algorithm is A-schedulable on m speed-b processors.

Algorithms for the federated scheduling of implicit-deadline sporadic DAG task systems were studied in [17]. A metric somewhat different from the speedup factor, the *capacity augmentation bound*, was used there:

Definition 2 (Capacity augmentation bound): Algorithm A for the federated scheduling of sporadic DAG task systems is said to have a *capacity augmentation bound* b if any task system τ satisfying both (i) $U_{\text{sum}}(\tau) \leq m$, and (ii) $\text{len}_i \leq D_i$ for each $\tau_i \in \tau$, is A-schedulable on m speed-b processors.

This research. As we have stated above, the federated scheduling of implicit-deadline sporadic DAG task systems was considered in [17]. In this paper we study federated scheduling for the more general *constrained*-deadline sporadic DAG task model. Although an argument can be made for the use of capacity augmentation bounds to quantify scheduling algorithms for implicit-deadline task systems, this is not a suitable metric for the analysis of constrained-deadline or arbitrary-deadline task systems. The reason why is illustrated in the following example.

Example 2: Consider a sporadic DAG task system consisting of the n sporadic DAG tasks $\tau_1, \tau_2, \ldots, \tau_n$, with the parameters of τ_i as follows:

- G_i consists of a single vertex v with WCET $e_v = 1$;
- $D_i = 1$; and
- $T_i = n$.

This task system τ has utilization $U_{\text{sum}}(\tau) = n \times \frac{1}{n}$ or 1, and each task τ_i satisfies $\text{len}_i \leq D_i$. However, it is only schedulable upon a processor of speed n. Hence as $n \to \infty$, a speedup of ∞ is necessary to schedule this system. ■

The example above illustrates that for any value of b, sporadic DAG task systems that are unschedulable without a capacity augmentation of b exist; hence, the capacity augmentation bound of any scheduling algorithm is necessarily zero. Since we are interested here in the scheduling of arbitrary-deadline task systems and this example reveals that capacity bounds cease to be a meaningful metric of scheduler effectiveness beyond the implicit-deadlines model, we will therefore stick to the more commonly-used speedup bounds to quantify the effectiveness of algorithms for scheduling sporadic DAG task systems. We present both previous results and the results derived in this paper within the common framework of speedup bounds in Section III below. A detailed description of a new federated scheduling algorithm for constrained-deadline sporadic DAG task systems is provided in Section IV.

III. The Current State of the Art

In this section we briefly summarize what is already known (or can be easily deduced from prior results in related topics) about the federated scheduling of sporadic DAG task systems.

Computational complexity. The problem we are seeking to solve is clearly highly intractable; indeed, it encapsulates two highly intractable sub-problems:

- Partitioning a given collection of tasks represented using the far simpler Liu and Layland model is already known to be NP-hard in the strong sense [13].
- Alternatively, determining whether a sporadic DAG task system consisting of just a single implicit-deadline sporadic DAG task is schedulable upon a given number m of processors is easily seen to be equivalent to the makespan minimization problem for preemptive scheduling of a set of precedence constrained jobs on identical processors. This problem is already known to be NP-hard in the strong sense and to remain so [15] even when the scheduler is given a speedup $(4/3 - \epsilon)$, for any $\epsilon > 0$.

Scheduling algorithms. For *implicit*-deadline sporadic DAG task systems, a federated scheduling algorithm was presented in [17], [18], and shown to have a capacity augmentation bound of 2. As shown in [16], a capacity augmentation bound b implies a speedup-bound bound b; hence, the algorithm of [17] also has a speedup bound of 2.

In Section IV below we will describe a two-phase algorithm titled Algorithm FEDCONS (represented in pseudo-code form in Figure 2) for the federated scheduling of any system of constrained-deadline sporadic DAG tasks. In the first phase, Algorithm MINPROCS will determine, for each high-density

FEDCONS(τ, m)

$\triangleright$ Constrained-deadline sporadic DAG task system τ is to be scheduled upon m processors.

1 $m_r \leftarrow m$ $\triangleright$ the remaining number of processors; initialized to m

2 **for** each $\tau_i \in \tau_{\text{high}}$

3 $m_i \leftarrow$ MINPROCS(τ_i, m_r) $\triangleright$ the minimum number of processors needed for τ_i – see Figure 3

4 **if** $(m_i > m_r)$ **return** FAILURE

5 $\sigma_i \leftarrow$ The schedule of τ_i's DAG on m_i processors generated by Graham's LS algorithm [12]

6 $m_r \leftarrow \left(m_r - m_i\right)$ $\triangleright$ update the remaining number of processors

end for

7 PARTITION(τ_{low}, m_r) $\triangleright$ See Figure 4

end

Fig. 2. Pseudo-code representation of the federated scheduling algorithm FEDCONS

task (i.e., a task with density ≥ 1), the number of processors to be devoted to this task. In the second phase, Algorithm PARTITION will partition the remaining (i.e., low-density) tasks upon the remaining processors. Algorithms MINPROCS and PARTITION will be shown to possess the following properties.

1) If a task can be scheduled in isolation by an optimal algorithm upon m unit-speed processors, Algorithm MINPROCS will schedule it upon m speed-2 processors.
2) If a given collection of low-density tasks can be optimally partitioned upon m unit-speed processors, then Algorithm PARTITION will partition it upon m speed-$\left(3 - \frac{1}{m}\right)$ processors.

Taken together, these results imply a speedup bound of $\left(3-\frac{1}{m}\right)$ for Algorithm FEDCONS.

A note. We point out an interesting distinction between the speedup results mentioned above for implicit-deadline and constrained-deadline task systems. The implicit-deadline federated scheduling algorithm of [17] can also be thought of as a two-phased algorithm: high-utilization tasks are first assigned processors, and the low-utilization tasks are partitioned next. The same Algorithm MINPROCS can be used for the first phase, again yielding a speedup bound of 2. However, the second phase – the partitioning of the low-utilization tasks upon the remaining processors – can be solved in polynomial time using the polynomial-time approximation scheme (PTAS) for partitioning Liu and Layland tasks that was derived in [13] (see also [9]). This means that the partitioning step can be done in polynomial time to have a speedup $(1 + \epsilon)$ for any constant $\epsilon > 0$. Hence the bottleneck for implicit-deadline systems is the high-utilization tasks — these are the tasks that are responsible for the speedup factor of 2. Stated somewhat differently, the increase in speedup factor from $(1 + \epsilon)$ to 2 is the price we pay for generalizing up from the Liu and Layland to the (implicit-deadline) sporadic DAG tasks model; the tasks for which the internal parallelism actually gets exploited are the ones that are responsible for the speedup factor of two.

For constrained-deadline sporadic DAG task systems, by contrast, the bottleneck step (i.e., the step with the worse speedup factor) is the partitioning step, not the step dealing with the high-density tasks. Thus in one sense, we can think of the generalization up to the sporadic DAG tasks model as being "for free" in that we are not incurring an additional penalty in terms of a poorer (larger) speedup factor.

IV. A FEDERATED SCHEDULING ALGORITHM

We now describe and discuss Algorithm FEDCONS, our algorithm for the federated scheduling of a constrained-deadline sporadic DAG task system τ upon a multiprocessor platform comprised of m unit-speed processors. Let $\tau_{\text{high}} \subseteq \tau$ denote the high-density tasks in τ (i.e., those tasks in τ with density ≥ 1). Let $\tau_{\text{low}} \stackrel{\text{def}}{=} (\tau \setminus \tau_{\text{high}})$ denote the low-density tasks in τ. In a first phase, Algorithm FEDCONS (pseudo-code in Figure 2) determines how many processors m_i to assign for the exclusive use of each high-density task τ_i, and also constructs a "template" schedule σ_i for a DAG-job of τ_i upon m_i processors under the assumption that each job executes for its specified WCET. The schedule σ_i is generated using Graham's List-Scheduling algorithm [12] (LS). In the second phase, the low-density tasks are partitioned amongst the remaining processors. Algorithm FEDCONS returns FAILURE if it either runs out of processors during the first phase, or fails to partition the low-density tasks upon the remaining processors during the second phase.

During run-time, jobs of the high-density tasks are executed upon the processors that are devoted to them as discussed below, while each shared processor is scheduled using preemptive uniprocessor Earliest Deadline First (EDF).

A. HIGH-DENSITY TASKS

Since the relative deadline of a constrained-deadline sporadic DAG is no greater than its period, all the jobs of one dag-job of the task must complete execution before the next dag-job is released. The problem of scheduling the sporadic DAG is therefore equivalent to determining whether each individual dag-job can be scheduled on the processors assigned to the task to complete within an interval of duration equal to the task's relative deadline parameter. The problem of scheduling a single constrained-deadline sporadic DAG upon a dedicated collection of processors is thus equivalent to the

makespan minimization problem for preemptive scheduling of a set of precedence constrained jobs on identical processors. It is known that Graham's *list scheduling* algorithm (LS) [12], which essentially constructs a work-conserving schedule by always executing an available job, if any are present, upon any available processor, has a speedup bound of $(2 - 1/m)$ for this problem. The procedure MINPROCS(τ_i, m_r) of Figure 3, called by FEDCONS for each high-utilization task τ_i, determines the minimum number of processors upon which LS generates a schedule for τ_i's DAG G_i that is of makespan no larger than D_i; if this number is greater than m_r then MINPROCS(τ_i, m_r) returns ∞. This schedule is stored as σ_i by FEDCONS; when a dag-job of τ_i arrives during run-time, σ_i is used as a lookup table to schedule the jobs of this dag-job.[2] It follows from the speedup bound of LS that

Lemma 1: If a particular constrained-deadline sporadic DAG task τ_i can be scheduled by an optimal scheduler to meet all its deadlines upon m_i unit-speed processors, then LS will schedule τ_i to meet all its deadlines upon m_i processors each of speed $(2 - 1/m_i)$.

B. LOW-DENSITY TASKS

In the partitioning step, multiple sporadic DAG tasks may be assigned to the same processor; however each task is assigned to a single processor. Since any intra-task parallelism cannot be exploited upon a single processor, we may ignore the internal structure of the DAG and simply represent such a task τ_i in the simpler *three-parameter sporadic* model [21]. A task τ_j in the three-parameter sporadic model is characterized by the three parameters C_j, D_j and T_j, representing respectively the worst-case execution time, relative deadline, and period parameters of the task. Such a task's jobs are assumed to possess no internal parallelism, but must be executed sequentially upon a single processor. For the purposes of partitioning the low-utilization sporadic DAG tasks, each such low-utilization sporadic DAG task $\tau_i = (G_i, D_i, T_i)$, can therefore be considered as equivalent to a three-parameter sporadic task with worst-case execution time equal to vol_i, relative deadline D_i, and period T_i.

[2]We point out that it is *not* safe to simply re-run LS during run-time – it was shown [11] that LS exhibits anomalous behavior in the sense that reducing the execution-times of jobs may increase the schedule length. Therefore, we choose to use the schedule σ_i as a lookup table during run-time, simply idling the processor for the remainder of the duration if a job executes for less than its WCET.

MINPROCS(τ_i, m_r)
1 **for** $\mu \leftarrow \lceil \delta_i \rceil$ **to** m_r **do**
2 Apply the List Scheduling algorithm [12] to construct a schedule for G_i on μ processors
3 **if** this schedule has makespan $\leq D_i$ **return** μ
end for
4 **return** ∞
end

Fig. 3. The procedure MINPROCS

PARTITION(τ_{low}, m_r)
▷ Without loss of generality, assume that $D_i \leq D_{i+1}$ for all i. $\tau(k)$ denotes the tasks assigned to the k'th processor; initially, $\tau(k) \leftarrow \varnothing$ for all k.
1 **for** $i \leftarrow 1$ **to** $|\tau_{\text{low}}|$
2 **for** $k \leftarrow 1$ **to** m_r
3 **if** $\left(D_i - \sum_{\tau_j \in \tau(k)} \text{DBF}^*(\tau_j, D_i)\right) \geq \text{vol}_i$ **then**
▷ assign τ_i to the k'th processor, and proceed
▷ to the next task
4 $\tau(k) \leftarrow \tau(k) \bigcup \{\tau_i\}$; **goto** line 6
5 **end** (of inner for loop)
6 **if** $(k > m_r)$ **return** FAILURE
7 **end** (of outer for loop)
end

Fig. 4. Pseudo-code for partitioning algorithm.

An algorithm for partitioning a specified collection of constrained-deadline sporadic three-parameter tasks upon a given multiprocessor platform was derived in [7]. This algorithm, presented in pseudo-code form in Figure 4, makes use of the DBF* approximation to the *demand bound function (*DBF*)* [2] of a three-parameter sporadic task, which we restate in terms of the parameters of a low-utilization sporadic DAG task:

$$\text{DBF}^*(\tau_i, t) = \begin{cases} 0, & \text{if } t < D_i \\ \text{vol}_i + u_i \times (t - D_i), & \text{otherwise} \end{cases} \quad (1)$$

The following result (re-stated here in terms of the parameters of a sporadic DAG task system) was derived in [7, Corollary 1, page 923]:

Lemma 2: If constrained-deadline sporadic DAG task system τ_{low} can be partitioned by an optimal algorithm on m_r identical processors each of a particular computing capacity, then PARTITION(τ_{low}, m_r) will successfully partition this system upon a platform comprised of m_r processors that are each $(3 - 1/m_r)$ times as fast as the original.

Combining Lemmas 1 and 2 and observing that the speedup bound of Lemma 2 is worse (larger) than that of Lemma 1, we immediately have

Theorem 1: If constrained-deadline sporadic DAG task system τ can be scheduled by an optimal federated scheduling algorithm on m identical processors each of a particular computing capacity, then FEDCONS(τ, m) will successfully schedule this system upon a platform comprised of m processors that are each $(3 - 1/m)$ times as fast as the original.

Proof Sketch: Consider a constrained-deadline sporadic DAG task system τ in which each task $\tau_i \in \tau$ has $\delta_i < 1$. Since there are no high-density tasks in τ, Procedure FEDCONS(τ, m) will partition the entire task system via the call to Procedure PARTITION (see Line 7 of the pseudocode of Figure 2).

For this task system τ, therefore, the performance of Procedure FEDCONS(τ, m) is dictated entirely by its performance in partitioning a constrained-deadline three-parameter sporadic task system, which is as stated in Lemma 2 above. ■

A note. The speedup bound of Theorem 1 is a conservative characterization of the efficacy of Algorithm FEDCONS. We have conducted schedulability experiments upon randomly-generated task systems to obtain a flavor of its more "typical" performance. Although we should not read too much into the results of these experiments since such results are necessarily deeply influenced by the manner in which we generate our task systems, the results do indicate that (as expected) the algorithm's performance is generally overwhelmingly better than implied by the conservative bound of Theorem 1.

V. SUMMARY AND CONCLUSIONS

Federated scheduling is a natural generalization of partitioned scheduling for tasks that exhibit internal parallelism and may need to execute upon multiple processors simultaneously. Prior work has considered the federated scheduling of implicit-deadline sporadic DAG tasks; here, we have obtained analogous results concerning the federated scheduling of systems of the more general constrained-deadline sporadic DAG tasks. Our worst-case bounds (Theorem 1) indicate that at least in terms of the speedup metric, there is no loss in going from the constrained-deadline three-parameter sporadic tasks model to the more general constrained-deadline sporadic DAG tasks model; schedulability experiments seem to imply that the worst-case bound of Theorem 1 is conservative.

A natural extension to this work would be to consider the federated scheduling of arbitrary-deadline sporadic DAG task systems. This is quite a bit more challenging than the problem considered in this paper, since a straightforward application of List Scheduling can no longer be used to obtain schedules for the high-density tasks.

ACKNOWLEDGEMENTS

This research was funded in part by NSF grants CNS 1016954, CNS 1115284, CNS 1218693, and CNS 1409175; and ARO grant W911NF-09-1-0535.

REFERENCES

[1] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic dag task systems," in *Proceedings of the 2012 26th Euromicro Conference on Real-Time Systems*, ser. ECRTS '14. Madrid (Spain): IEEE Computer Society Press, 2014.

[2] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of the 11th Real-Time Systems Symposium*. Orlando, Florida: IEEE Computer Society Press, 1990, pp. 182–190.

[3] S. Baruah, "A general model for recurring real-time tasks," in *Proceedings of the Real-Time Systems Symposium*. Madrid, Spain: IEEE Computer Society Press, December 1998, pp. 114–122.

[4] ——, "Improved multiprocessor global schedulability analysis of sporadic dag task systems," in *Proceedings of the 2014 26th Euromicro Conference on Real-Time Systems*, ser. ECRTS '14. Madrid (Spain): IEEE Computer Society Press, 2014, pp. 97–105.

[5] 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*, ser. RTSS 2012, San Juan, Puerto Rico, 2012, pp. 63–72.

[6] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Systems: The International Journal of Time-Critical Computing*, vol. 17, no. 1, pp. 5–22, July 1999.

[7] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 918–923, July 2006.

[8] 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*, ser. ECRTS '13, Paris (France), 2013, pp. 225–233.

[9] B. Chattopadhyay and S. Baruah, "A lookup-table driven approach to partitioned scheduling," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. Chicago: IEEE Computer Society Press, 2011.

[10] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Information Processing Letters*, vol. 106, no. 5, pp. 180–187, 2008.

[11] R. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, pp. 1563–1581, 1966.

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

[13] D. Hochbaum and D. Shmoys, "Using dual approximation algorithms for scheduling problems: Theoretical and practical results," *Journal of the ACM*, vol. 34, no. 1, pp. 144–162, Jan. 1987.

[14] 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.

[15] 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.

[16] 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*, ser. ECRTS '13, Paris (France), 2013, pp. 3–13.

[17] J. Li, A. Saifullah, K. Agrawal, C. Gill, and C. Lu, "Analysis of federated and global scheduling for parallel real-time tasks," in *Proceedings of the 2012 26th Euromicro Conference on Real-Time Systems*, ser. ECRTS '14. Madrid (Spain): IEEE Computer Society Press, 2014.

[18] ——, "Capacity augmentation bound of federated scheduling for parallel dag tasks," Washington University in St Louis, Tech. Rep. WUCSE-2014-44, 2014.

[19] 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.

[20] G. Manimaran, C. S. R. Murthy, and K. Ramamritham, "A new approach for scheduling of parallelizable tasks in real-time multiprocessor systems," *REAL-TIME SYSTEMS*, vol. 15, pp. 39–60, 1998.

[21] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983, available as Technical Report No. MIT/LCS/TR-297.

[22] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," in *Proceedings of the 17th Real-Time Systems Symposium*. Washington, DC: IEEE Computer Society Press, 1996.

[23] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, Nov 2011, pp. 217–226.

[24] M. Stigge, "Real-time workload models: Expressiveness vs. analysis efficiency," Ph.D. dissertation, Ph.D. thesis, Uppsala University, 2014.

[25] 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.