2017 IEEE Real-Time Systems Symposium

Global EDF-based scheduling of multiple independent synchronous dataflow graphs

Abhishek Singh
The University of North Carolina at Chapel Hill
abh@cs.unc.edu

Sanjoy Baruah
Washington University in St. Louis
baruah@wustl.edu

***Abstract*—The global scheduling of systems that can be modeled as collections of multiple independent recurrent real-time tasks, each represented as a synchronous dataflow graph (SDFG), upon an identical multiprocessor platform is considered. An EDF-based scheduling algorithm is proved optimal under the speedup factor metric, and a speedup-optimal sufficient schedulability test is derived.**

***Keywords*—SDF Graphs; constrained-deadline recurrent tasks; global multiprocessor scheduling; speedup-optimal scheduling; pseudo-polynomial time schedulability analysis.**

I. Introduction

The research discussed in this manuscript arose out of interactions with colleagues in the telecommunication industry who shared with us some scheduling problems arising in the design of base-stations for wireless cellular communication systems. A base-station should be able to handle a certain number of connections. For each handled connection, streams of data packets arrive at the base-station and go through a number of stages of data-flow processing — these stages may be different for different connections. A specified minimum duration is assumed between the arrival of successive data packets of the same connection, and the processing of the packet is required to complete within a specified duration of its arrival.

It is natural to model such processing requirements using *sporadic task models* of the kind that have been very widely studied in the real-time scheduling literature, with the minimum inter-arrival duration between successive data packets modeled by the "period" parameter, and the duration allowed for the processing of each packet by the "relative deadline" parameter. In the telecom applications we were dealing with the actual processing of the packets were represented using the Synchronous Data Flow Graph (SDFG) [13] abstraction; there does not appear to be a straight-forward means of directly modeling such processing requirements using the concepts and terminology of real-time scheduling theory. Although the SDFG abstraction has been studied for decades, the run-time scheduling of computational workloads that are represented in the SDFG model has traditionally been done via static scheduling methods (e.g., [14], [12]) in which scheduling tables are determined prior to run-time and these pre-computed tables are used for making run-time scheduling decisions. As embedded streaming applications become increasingly more computation-intensive and efficiency of implementation becomes an increasing concern, however, efforts are being made (e.g., [8], [2], [3], [1], [15], [10], [11] – this is not an exhaustive list) to explore the use of dynamic scheduling approaches, of the kind that are studied in the real-time scheduling theory community, in order to obtain more resource-efficient implementations of systems that are represented using the SDFG model. To our knowledge, none of these prior approaches claim optimality or provable quantitative bounds on deviation from optimality; indeed, it is fairly easy to construct example instances in which each prior approach will result in implementations that make arbitrarily inefficient use of platform computing resources.

Motivation. The long-term objective of our research is to investigate the applicability of the concepts, techniques, methodologies, and results of real-time scheduling theory to the analysis of real-time workloads that are represented using the SDFG model. We believe that there is plenty of opportunity here: real-time scheduling theory has made tremendous progress in recent years, but this progress has, by and large, remained focused upon the workload models popular within the community. Meanwhile data-flow models such as SDFG are finding increasing use in many embedded application domains, but research on these models is only now beginning to address the issue of enhancing implementation efficiency via dynamic scheduling. This offers us in the real-time scheduling theory community an opportunity to demonstrate the usefulness and applicability of our research endeavors, and provides us with a rich source of interesting new problems that are of immediate interest outside the real-time scheduling theory community.

This research. Sporadic tasks in which the processing requirements are represented using the SDFG model have been called *sporadic real-time SDFG tasks*. An algorithm based upon the Earliest Deadline First scheduling strategy (EDF) was presented in [19] for scheduling multiple independent sporadic real-time SDFG tasks, and proved to be uniprocessor optimal in the following sense: if a particular real-time system modeled as a collection of multiple independent sporadic real-time SDFGs could be scheduled upon a preemptive uniprocessor by any algorithm to always meet all deadlines, then the algorithm in [19] also guarantees to always meet all deadlines when scheduling this system. An exact schedulability test for this algorithm was also presented; a more efficient exact schedulability test, with run-time polynomial in the representation of the system being scheduled, was subsequently presented in [20] (the test in [19] had exponential worst-case running time).

In this paper, we consider the global scheduling of systems of multiple independent sporadic real-time SDFGs upon preemptive multiprocessor platforms. We generalize the results of [19], [20] by

1) showing that (the global multiprocessor extension to)

1052-8725/17/31.00 ©2017 IEEE
DOI 10.1109/RTSS.2017.00036

the algorithm of [19] is *speedup optimal* for EDF-based scheduling, by showing that it has a speedup bound of $(2 - \frac{1}{m})$ upon an m-processor platform, while no EDF-based algorithm may have a smaller speedup bound; and
2) deriving a speedup-optimal sufficient schedulability test for this algorithm.

We thus show that the results for recurrent task systems represented using the sporadic real-time SDFG model mirror the results for recurrent task systems represented using traditional real-time scheduling models such as 3-parameter sporadic tasks [16] and sporadic DAG tasks [6] — for both these models, EDF-scheduling has previously been proved optimal upon preemptive uniprocessors and shown to have a speedup bound of $(2 - \frac{1}{m})$ upon preemptive multiprocessors, and speedup-optimal sufficient schedulability-analysis tests upon both uni- and multi-processors are known.

Organization. The remainder of this paper is organized in the following manner. The sporadic real-time SDFG model is described in Section II. A closely related model, the sporadic DAG model, is discussed in Section III; here we also explore similarities and differences between the two models. Section IV extends some concepts concerning sporadic DAG tasks to derive a speedup bound for the global scheduling of multiple independent sporadic real-time SDFGs on a shared platform; Section V derives an optimal schedulability test.

II. The sporadic real-time SDFG Model

A dataflow graph is a directed graph[1] in which the vertices (called *actors*) represent computation and edges (called *channels*) represent FIFO queues that direct data (called *tokens*) from the output of one computation to the input of another. Actors *consume* tokens from their input channels, perform computations upon them (this is referred to as a *firing* of the actor) and *produce* tokens on their output channels. Channels may contain *initial tokens* (also known, for historical reasons, as *delays*) — these represent data that populate the FIFO queues prior to run-time.

A *synchronous dataflow graph (SDFG)* is a dataflow graph with the additional constraint that the number of initial tokens on each channel, as well as the number of tokens produced (consumed, respectively) by each actor on each of its outgoing (incoming, resp.) channels upon a firing of the actor, is a known constant. More formally, an SDFG G is represented by a 5-tuple of parameters: $G = \langle V, E, \text{PROD}, \text{CONS}, \text{DELAY} \rangle$, where

- V denotes the actors.
- $E \subseteq V \times V$ denotes the channels.
  For channel $e = (u, v)$, we refer to u as $\text{TAIL}(e)$ and v as $\text{HEAD}(e)$. We assume that $\text{TAIL}(e) \neq \text{HEAD}(e)$ for any e; i.e., no channel leads from an actor back to itself.

[1]Dataflow models often also allow for multiple edges between a pair of vertices, and self-loops, which are edges leading from a vertex back to itself. These semantically relevant features do not impact schedulability, and are we therefore ignore them in this paper — our results are easily extended to deal with multiple edges and self-loops.

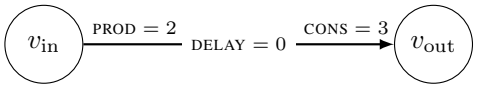


Fig. 1: **An example SDFG. Each vertex is labeled with the name of the actor it represents; edge $(v_{\text{in}}, v_{\text{out}})$ denotes a channel leading from v_{in} to v_{out}. The channel is labeled with its produce and consume rates, and its delay.**

- $\text{PROD} : E \to \mathbb{N}_{>0}$. For each $e \in E$, $\text{PROD}(e)$ tokens are added to channel e each time the actor $\text{TAIL}(e)$ fires.
- $\text{CONS} : E \to \mathbb{N}_{>0}$. For each $e \in E$, $\text{CONS}(e)$ tokens are removed from channel e each time the actor $\text{HEAD}(e)$ fires.
- $\text{DELAY} : E \to \mathbb{N}_{\geq 0}$. For each $e \in E$, there are $\text{DELAY}(e)$ initial tokens on channel e.

The number of tokens in each channel in E determines the state or configuration of the SDFG $G = \langle V, E, \text{PROD}, \text{CONS}, \text{DELAY} \rangle$. For a particular configuration of G, an actor $v \in V$ is said to be *enabled* if each channel $e \in E$ for which $\text{HEAD}(e) = v$ contains at least $\text{CONS}(e)$ tokens. An enabled actor v may *fire*; doing so changes the configuration of the SDFG in the following manner: $\text{CONS}(e)$ tokens are removed from each channel $e \in E$ for which $\text{HEAD}(e) = v$, and $\text{PROD}(e)$ tokens are added to each channel $e \in E$ for which $\text{TAIL}(e) = v$. Algorithms have been developed [14], [13] for determining, for a given SDFG, whether sequences of firings could lead to *deadlock* — a configuration in which no actor is enabled, or to *buffer overflow* — the number of tokens in a channel growing without bound.[2] In the remainder of this paper we will assume that the SDFGs we deal with have been *a priori* verified to be deadlock-free and not subject to buffer overflow.

Definition 1 (Repetitions vector; Iteration)**.** *The **repetitions vector** for an SDFG G is the unique smallest positive integer vector $\mathbf{q} : V \to \mathbb{N}$ (not equal to $\mathbf{0}$) such that if each actor v fires $\mathbf{q}(v)$ times, then the total number of tokens in each channel is unchanged. An **iteration** is a set of actor firings with as many firings as the repetitions vector entry for each actor.* □

A ***homogeneous*** SDFG is an SDFG in which all the PROD and CONS rates are equal to one. Algorithms are known [14] for converting any SDFG into an equivalent homogeneous one (which may, however, be of size exponential to the original SDFG – such an exponential blow-up in size is known to be unavoidable in the worst case).

A. Incorporating real-time considerations

As initially defined [14], [13], SDFGs do not deal with real time: "SDF is an untimed model of computation. All actors under SDF consume input tokens, perform their computation

[2]In addition to requiring that each buffer be of finite size, many SDFG scheduling algorithms seek to minimize the maximum number of tokens each buffer will need to hold. For simplicity we do not consider buffer-size minimization here, simply requiring that they be of finite size and leaving as future work the problem of determining the minimum sizes needed.

and produce outputs in one atomic operation." [17, page 53]. Real-time modeling capabilities were added by incorporating the notions of (i) *latency* between the executions of different actors [9]; and (ii) the *response time* to external triggering events that may occur recurrently in a sporadic manner [19]. To account for execution times, an additional parameter $\big(\text{WCET} : V \rightarrow \mathbb{N}_{\geq 0}\big)$ was added to the specification of an SDFG, with the interpretation that for each $v \in V$, $\text{WCET}(v)$ is the worst-case execution time of a (single) firing of actor v. In order to explicitly represent real-time responsiveness to recurrent external triggering events, the SDFG model was extended as follows: for each SDFG, we are required to additionally specify

1) A single *input actor* v_{in} and a single *output actor* v_{out}. External tokens are assumed to arrive at v_{in}.
2) A *period* parameter, denoting the minimum duration between successive arrivals of external tokens at v_{in}.
3) A *relative deadline* parameter, denoting the maximum duration that may elapse between the arrival of an external token at v_{in} and the completion of the "*corresponding"* execution of the output actor v_{out} (this notion of correspondence is formalized in Definition 2 below).

Suppose that the simple SDFG shown in Figure 1 represents a real-time sporadic SDFG (i.e., in this extended model). Suppose that the first external input token arrives at actor v_{in} at some time-instant, thereby causing v_{in} to fire. Observe that since the produce rate of the channel leading from v_{in} to v_{out} is two while the consume rate is three, at least two firings of v_{in} must occur before actor v_{out} may fire for the first time. But since the period of the SDFG denotes only a *lower* bound on the duration between the arrival of successive external input tokens, we cannot provide an upper bound upon the time-instant at which actor v_{out} is enabled – this depends upon when the second external input token arrives at actor v_{in}. It is therefore not meaningful to discuss the latency of the response to the first external input token, since the response will be triggered by not the first, but the second external input token. Ghamarian et al. [9] sidestepped the dilemma that this poses, by arguing that an entire iteration (see Definition 1) of an SDFG should be thought of as representing a single logical chunk of computation. Therefore it is not meaningful to consider the arrivals of external input tokens at the input actor, and firings of the output actor, within an iteration; instead, we should only consider the delay between the arrival of an external input token that initiates the first firing of the input actor within an iteration, and the completion of the execution of the last firing of the output actor during that iteration. They therefore proposed [9, page 191] that any SDFG be preprocessed by "add[ing] an explicit source actor to the [input actor] and a destination actor to the [output actor], each of which fires by construction exactly once in every iteration of the graph. If an SDFG already has meaningful input and output actors with repetition vector entries of one, these actors can function as source and destination and no actors need to be added." The precise details of such construction are elaborated upon in [9]. Henceforth, we will assume that our SDFGs have been pre-processed in this manner, and that as a consequence, we have SDFGs with designated input and output actors that are guaranteed to execute exactly once per iteration.

An additional factor that must be taken into account arises from the tokens that populate each channel initially, as specified by the DELAY parameters. There are $\text{DELAY}(e)$ such tokens on each $e \in E$; since each $\text{DELAY}(e)$ is finite and since we require that each actor be reachable from v_{in}, any actor can fire at most a finite number of times prior to v_{in} firing for the first time. The *dependency distance* denotes the maximum number of times the output actor v_{out} can fire before exhausting the initially-supplied tokens:

Definition 2 (Dependency Distance δ [21]; Correspondence)**.** *Due to the initial distribution of tokens on the channels specified by* DELAY*, v_{out} can fire some δ times before the first firing of v_{in}. The number δ is called the dependency distance.*

For any $k \in \mathbb{N}$, the k-th firing of v_{in} is said to correspond to *the $(k+\delta)$-th firing of v_{out}, where δ is the dependency distance.* □

Summarizing the model. We will refer to the recurrent task model obtained by making all the enhancements discussed above to the "traditional" SDFG model as the *sporadic real-time SDFG* model. A task in this model is specified by the following parameters:

$$G \stackrel{\text{def}}{=} \Big\langle V, E, \text{PROD}, \text{CONS}, \text{DELAY}, \text{WCET}, v_{\text{in}}, v_{\text{out}}, D, T \Big\rangle \tag{1}$$

with

- V, E, PROD, CONS, and DELAY as specified for traditional SDFGs;
- $\text{WCET} : V \rightarrow \mathbb{N}_{\geq 0}$ specifying the worst-case execution times of the actors;
- Actors $v_{\text{in}} \in V$ and $v_{\text{out}} \in V$ being specified as the unique input and output actor, respectively; and
- $D \in \mathbb{N}$ and $T \in \mathbb{N}$ specifying the relative deadline and period parameters of this sporadic real-time SDFG task.

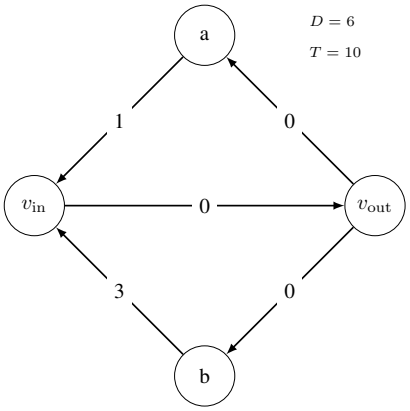
Additionally, we assume that the SDFG has been validated to be deadlock-free and free from buffer overflow, and to have the repetition rates for the input and output actors equal to one: $\mathbf{q}(v_{\text{in}}) = \mathbf{q}(v_{\text{out}}) = 1$.

In this paper, we restrict our attention to sporadic real-time SDFGs possessing the additional property that their relative deadline parameter is no greater than their period: $D \leq T$ (recurrent tasks possessing this property are often referred to as *constrained deadline* tasks in the real-time scheduling literature).

We define the *utilization* $U(G)$ of such a sporadic real-time SDFG as follows:

$$U(G) = \Big(\sum_{v \in V} \big(\text{WCET}(v) \times \mathbf{q}(v)\big)\Big)/T \tag{2}$$

An example sporadic real-time HSDFG comprising four actors and five channels between these actors is depicted in



actor	v_{in}	v_{out}	a	b
WCET	1	1	1	2

Fig. 2: **An example sporadic real-time HSDFG; the channels are labeled with the number of initial tokens.**

Figure 2. Since this is a homogeneous SDFG, all the produce and consume rates are equal to one. The worst-case execution requirement parameters, relative deadline, and period are as specified in the diagram (observe that D is indeed $\leq T$).

Some prior results from [19], [20]. An EDF-based algorithm was derived in [19] for scheduling multiple independent sporadic real-time SDFGs optimally upon a preemptive uniprocessor. As is fairly standard in preemptive uniprocessor EDF schedulability analysis, this algorithm quantifies the computational demand of each individual task via the *demand bound function* [4]. For any sporadic real-time SDFG G and any positive real number t, let $\text{DBF}(G, t)$ denote the maximum cumulative execution requirement of all actors that can be generated by G to have both their arrival times and their deadlines within a contiguous interval of length t. With respect to this particular sporadic real-time SDFG G, let $k(t)$ denote the following function:

$$k(t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t-D}{T} \right\rfloor + 1\right)\right) \tag{3}$$

It is evident that over any contiguous time-interval of duration t there may be at most $k(t)$ external input tokens arriving at v_{in} for which the corresponding firings of v_{out} must occur within the interval. Since each arrival of an external input token at v_{in} triggers one iteration of G, an upper bound for $\text{DBF}(G, t)$ may be obtained by simply assuming that each actor a fires a total of $\mathbf{q}(a)$ times during each such iteration, thereby obtaining a bound of

$$k(t) \times \sum_{v \in V} \big(\mathbf{q}(v)\text{WCET}(v)\big) \tag{4}$$

This bound, while safe, may not be tight – the presence of initial tokens on some of the channels (as represented by the $\text{DELAY}(c)$ values) means that not all firings of all actors need take place, as illustrated in the following example.

Example 1. Consider the example HSDFG of Figure 2 and consider a value of t satisfying $D < t < T + D$, so that $k(t)$ evaluates to 1 by Equation 3. Hence Expression 4 is computed to have a value of

$$\Big(\text{WCET}(v_{\text{in}}) + \text{WCET}(v_{\text{out}}) + \text{WCET}(a) + \text{WCET}(b)\Big) = 5$$

However, the reader may verify that the presence of initial tokens ensures that v_{out} is able to fire even if actors a and b do *not* fire. Hence over such a t, $\text{DBF}(G, t)$ equals

$$\Big(\text{WCET}(v_{\text{in}}) + \text{WCET}(v_{\text{out}})\Big) = 2,$$

which is smaller than the value computed by Expression 4. □

The EDF-based scheduling algorithm of [19] exploits the observation illustrated in the example above to reduce the DBF. It first computes, prior to run-time, a *skip vector* $\mathbf{s}(\cdot)$ of non-negative integers with $|V|$ components, which represents the maximum number of firings of each actor that may be "skipped" (i.e., postponed) as a consequence of the presence of initial tokens on the channels. These skip vector values are used during run-time to assign deadlines to the execution of individual actors; the precise manner in which this is done is detailed in [19]. For instance in the example above, the computed skip vector values for the HSDF of Figure 2 turn out to be

$$\mathbf{s}(v_{\text{in}}) = \mathbf{s}(v_{\text{out}}) = 0, \mathbf{s}(a) = 1, \text{ and } \mathbf{s}(b) = 3\ . \tag{5}$$

During run-time if an external token arrives at v_{in} at time-instant t_o, an instance of each actor is released at t_o with deadline at time-instants $t_o + D$ for v_{in} and v_{out}, $(t_o + D + T)$ for a, and $(t_o + D + 3T)$ for b. The correctness of this strategy is formally proved in [19], and algorithms for computing the skip vector values in polynomial time are presented in [19] for HSDFGs, and in [20] for general (i.e., not necessarily homogeneous) SDFGs.

III. The sporadic DAG task model

In the sporadic DAG tasks model [6], each task is specified by a tuple (G_i, D_i, T_i), where $G_i = (V_i, E_i)$ is a vertex-weighted directed acyclic graph, and D_i and T_i are positive integers. Each vertex $v \in V_i$ of the DAG corresponds to a sequential job, and is characterized by a worst-case execution time (WCET) e_v. Each edge 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 if there are processors available for them to do so. We say the task G_i releases a *dag-job* at time-instant t when it becomes available for execution. When this happens, we assume that all $|V_i|$ of the jobs are released and become available for execution, subject to the precedence constraints. During any given run the task may release an unbounded sequence of dag-jobs; all $|V_i|$ jobs that are released at some time-instant t must complete execution by time-instant $t + D_i$. A minimum interval of

duration T_i must elapse between successive releases of dag-jobs.

The scheduling of systems of multiple independent sporadic DAG tasks has been studied extensively in the real-time scheduling literature (see, e.g., [5, Chapter 21] for a textbook survey of some results). It is known, for example, that global EDF has a speedup bound of $(2 - 1/m)$ in scheduling such systems upon m-processor platforms, and pseudo-polynomial EDF schedulability analysis algorithms have been derived for bounded-utilization systems – please see [5, Chapter 21] for details.

A. Reducing sporadic real-time SDFGs to DAG tasks

It is evident that sporadic real-time SDFGs are very similar to sporadic DAG tasks. Given any SDFG, we can first convert it to a homogeneous SDFG (HSDFG) – as previously stated, algorithms are known [14] for doing so. Once this is done, we can think of the firing of each actor of the HSDFG as corresponding to the execution of a particular job of the sporadic DAG task, and the channels of the HSDFG as representing precedence constraints between such jobs. Hence the arrival of an external token at the input actor of the HSDFG corresponds to the release of a dag-job of the sporadic DAG task; one iteration of the HSDFG corresponds to the execution of an entire dag-job.

In the absence of initial tokens (DELAYs), this correspondence between sporadic real-time SDFGs and sporadic real-time DAG tasks is exact, and schedulability results for sporadic DAG task systems are directly applicable to systems of sporadic real-time SDFGs. However, recall that the presence of initial tokens allows for some actor executions to be *skipped* during an iteration (or rather, to be postponed to future iterations). Such postponement of the execution of individual jobs is not supported in the sporadic DAG tasks model, and hence the results for sporadic DAG tasks cannot be directly applied to the execution of sporadic real-time SDFGs. We illustrate via an example.

Example 2. Recall (Expression 5) that the skip vector values for the example sporadic real-time HSDFG depicted in Figure 2 are as follows: $\mathbf{s}(v_{\text{in}}) = \mathbf{s}(v_{\text{out}}) = 0; \mathbf{s}(a) = 1$; and $\mathbf{s}(b) = 3$. In Figure 3, the dependencies amongst actor executions are shown for the first six iterations of this sporadic real-time HSDFG, assuming that external tokens arrive at v_{in} exactly $T = 10$ time units apart. (For those reading this manuscript on a color medium, alternate iterations are depicted in different colors – red and blue.) Notice that this figure is *not* equivalent to simply replicating six copies of the HSDFG of Figure 2, each representing one iteration, with successive copies separated by a duration $T = 10$ along the time axis. Instead, *the channels from the actors a and b to the actor v_{in} within each iteration have been replaced by precedence constraints leading from a to the v_{in} of the next iteration, and from b to the v_{in} three iterations later.* □

As illustrated in the example above, the presence of initial tokens in an SDFG means that schedulability analysis algorithms that were developed for sporadic DAG task systems are not directly applicable to the sporadic real-time SDFG model. However, some novel concepts that were developed to enable analysis of sporadic DAG task systems may be adapted to the analysis of systems of sporadic real-time SDFGs as well. In Section III-B below we discuss one particularly important such concept.

B. Well-formed collections of jobs

The concept of a *normal collection of jobs* was introduced in [7]; here, we propose a generalization:

Definition 3. *A* collection of jobs J *is a set of jobs that are revealed over time, with a job $j \in J$ becoming known upon its release time. Each job $j \in J$ is characterized by a release time $r_j \in \mathbb{N}_0$, an absolute deadline $d_j \in \mathbb{N}$, an execution time $e_j \in \mathbb{N}$, and a set of predecessor jobs J_j which are exactly the jobs which have to be finished before j can begin to execute.*

A collection of jobs J is said to be a ***well-formed*** *collection of jobs if it satisfies the additional property that for every predecessor job j of each job k, $d_j \leq d_k$.*

(This generalizes the concept of *normal collection* [7], for which it is required that additionally, $r_i = r_j$.)

The following generalization of a technical lemma from [7] provides a very useful characterization of global EDF schedules for well-formed collections of jobs:

Lemma 1. *Consider a well-formed collection of jobs J. Let $m \in \mathbb{N}$ denote any positive integer, and $\alpha \in \mathbb{R}_{\geq 1}$ any constant that is ≥ 1.* ***If*** *there exists some schedule meeting all deadlines for J upon unit-speed processors which, for any time interval I, completes no more than $(\alpha m - m + 1) \cdot |I|$ units of execution within I,* ***then*** *J is global-EDF schedulable on m speed-α processors.*

Proof: In the appendix. □

We can use Lemma 1 to derive a speedup bound for the global EDF scheduling of well-formed collections of jobs, as follows. Consider any instance J that is feasible upon m unit-speed processors; such feasibility immediately implies the existence of a valid schedule which completes at most $m \cdot |I|$ units of work in any interval I. Let us consider $\alpha \leftarrow (2 - 1/m)$; for this value of α, we have

$$(\alpha m - m + 1) \cdot |I| = (2m - 1 - m + 1) \cdot |I| = m|I|.$$

The speedup result in Lemma 2 immediately follows:

Lemma 2. *Any well-formed collection of jobs that is feasible on m unit-speed processors is EDF-schedulable on m processors each of speed $(2 - 1/m)$.*

The bound of Lemma 2 is tight: examples are known [18], even without precedence constraints (and hence trivially well-formed), of feasible collections of jobs that are not EDF-schedulable unless the speedup is at least $(2 - 1/m)$,

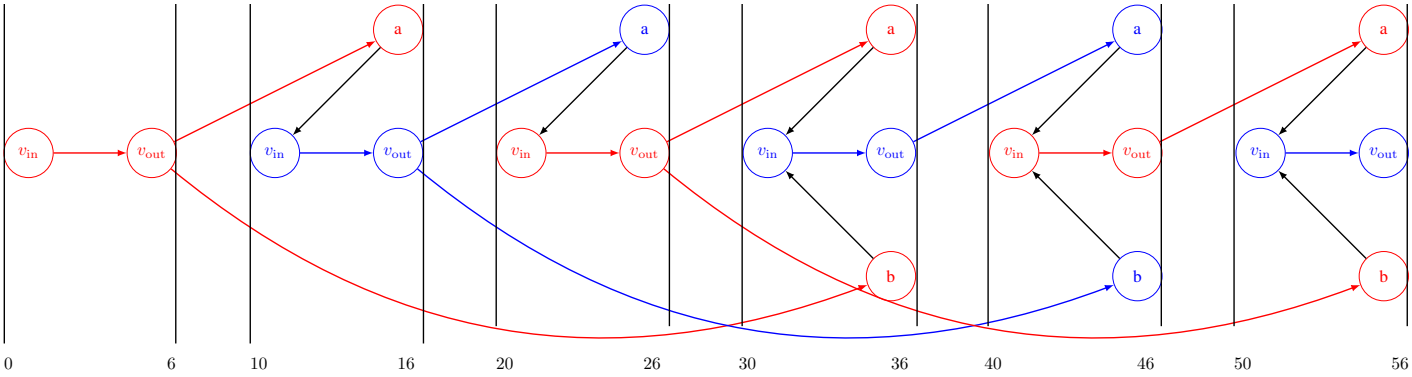


Fig. 3: Illustrating the dependencies between actor executions, for six iterations of the HSDFG depicted in Figure 2.

IV. A SPEEDUP BOUND FOR REAL-TIME SPORADIC SDFGS

As we had stated in Section II, the EDF-based scheduling algorithm of [19] for sporadic real-time SDFG tasks computes, for a given task G, the skip vector values for all the actors in G, and uses these skip vector values during run-time to assign deadlines to the executions of individual actors. We now describe this process in greater detail.

Given any sporadic real-time SDFG with period T and relative deadline D, consider the sporadic real-time homogeneous SDFG (HSDFG) that is equivalent to it. Let us refer to the execution of an actor as a job. The arrival of an external token at the input actor of the SDFG at some time-instant t_o results in the release of one job corresponding to each actor in the equivalent HSDFG, each with a release time t_o. The relative deadline assigned to each job, and the precedence constraints between the jobs, are determined based upon the skip vector values that are computed for the actors[3] and the initial tokens (DELAYS) on the edges. Specifically

- For each actor v, the corresponding job is assigned a deadline equal to $\big(t_o + \mathbf{s}(v) \times T + D\big)$; and
- For each channel (u, v) leading from some actor u to some actor v, a precedence constraint is added from the job corresponding to u to the job corresponding to the DELAY(u, v)'th-next invocation of actor v.

We illustrate this in the example below.

Example 3. Consider once again the dependencies amongst the actor executions for the first six iterations of the example sporadic real-time HSDFG of Figure 2 that are depicted in Figure 3.

Since $\mathbf{s}(v_{\text{in}}) = \mathbf{s}(v_{\text{out}}) = 0$,

- each job corresponding to an execution of v_{in} and v_{out} is assigned a deadline $D = 6$ time units after its release.
- the channels $(v_{\text{in}}, v_{\text{out}})$, (v_{out}, a), and (v_{out}, b) all result in precedence constraints between jobs of the same iteration.

Since $\mathbf{s}(a) = 1$,

- each job corresponding to an execution of a is assigned a deadline $(T + D) = 16$ time units after its release.
- the channel (a, v_{in}) results in a precedence from a to the v_{in} generated in the next iteration.

Since $\mathbf{s}(b) = 3$,

- each job corresponding to an execution of b is assigned a deadline $(3T + D) = 36$ time units after its release.
- the channel (b, v_{in}) results in a precedence from b to the v_{in} that is generated three iterations later.

□

Above we saw how we can represent an execution of the SDFG G by a collection of precedence-constrained jobs. Unlike DAGs, a sequence of precedence-constrained jobs for an SDFG may have unbounded length. For instance, in Figure 3, $(v_{\text{in}}, v_{\text{out}}, a, v_{\text{in}}, v_{\text{out}}, a, v_{\text{in}}, v_{\text{out}}, \ldots)$ is such a sequence. For G to be feasible upon a platform comprising speed-s processors, it is clearly necessary (albeit not sufficient) that the WCETs of each sequence of precedence-constrained jobs not exceed the deadline of the job at the end of the sequence relative to the release of the first job of the sequence; this is formally stated in the following lemma.

Lemma 3. *In order that sporadic real-time SDFG G be feasible upon a platform comprising speed-s processors, it is necessary that the sum of the WCETs of each sequence of precedence-constrained jobs not exceed the deadline of the job at the end of the sequence relative to the release of the first job of the sequence, times s.* □

It can be shown that for a given sporadic real-time SDFG G, determining whether G satisfies the condition of Lemma 3 may be accomplished by first transforming G to an equivalent HSDFG, and then checking this condition in time pseudo-polynomial in the representation of this HSDFG.

We now relate systems of sporadic real-time SDFGs to well-formed collections of jobs (Definition 3):

[3]The algorithm of [19] does not explicitly convert the SDFG to an equivalent HSDFG (such a conversion could take exponential time). Hence, the skip vector values of actors in the HSDFG are not explicitly computed; instead, the skip vector values of actors in the original SDFG are computed, and those for actors in the equivalent HSDFG deduced from these computed values.

Lemma 4. *For any system of independent sporadic real-time SDFGs, the jobs corresponding to the execution of their actors that are generated by the EDF-based scheduling algorithm of [19], in response to any legal sequence of arrivals of external tokens at their input actors, constitute a well-formed collection of jobs.*

Proof: Since the different sporadic real-time SDFGs are assumed to be independent, there are no precedence constraints between jobs of different SDFGs. It remains to show that jobs generated by an individual sporadic real-time SDFG satisfy the characterizing property of well-formed collections of jobs.

Given any sporadic real-time SDFG and any legal arrival sequence of external tokens at its input actor, transform the SDFG into an equivalent HSDFG and consider the jobs corresponding to the execution of its actors as defined above (and illustrated in Example 3). It is evident that these jobs constitute a well-formed collection of jobs (Definition 3), since if there is a precedence constraint from a job j in a particular iteration to the job j' that is ℓ iterations later ($\ell \geq 0$), the

- the deadline of job j is $(\ell \times T + D)$ time units after its release, while
- the deadline of job j' is $\geq (\ell \times T + D)$ time units after the release of the j'th job.

□

By combining Lemma 4 with Lemma 2, we immediately have the following speedup result for sporadic real-time SDFGs:

Theorem 1. *Any system of independent sporadic real-time SDFGs that is feasible on m unit-speed processors is schedulable on m processors each of speed $(2 - 1/m)$ by the EDF-based algorithm in [19].*

V. A SPEEDUP-OPTIMAL SCHEDULABILITY TEST

In order to derive a schedulability test for systems of sporadic DAG tasks, Bonifaci et al. [7] introduced the notion of a *work function* as a generalization to the concept of demand bound function. This important notion is easily adapted to sporadic real-time SDFGs; we do so in Section V-A below. Then in Section V-B we show how this work function may be used to form the basis for an effective schedulability test for systems of sporadic real-time SDFGs.

A. A work function for sporadic real-time SDFGs

Let s denote any positive real number. For any collection of jobs J, 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. (Figure 4 depicts the schedule $S_\infty(J, 0.5)$, for J representing the jobs that correspond to the firings of actors of the sporadic real-time HSDFG shown in Figure 2, when input tokens arrive at vertex v_{in} at time-instants 0 and 10.) The work function for sporadic real-time SDFGs is defined as follows:

- For any interval I, let $\mathsf{work}(J, I, s)$ 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.

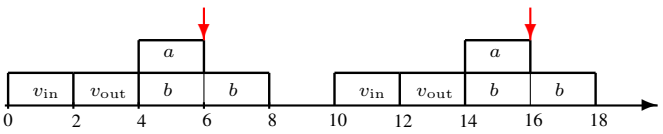


Fig. 4: The schedule $S_\infty(J, \frac{1}{2})$, where J is the collection of jobs corresponding to actor firings when input tokens arrive at time-instants 0 and 10 for the task shown in Figure 2.

  Observe that since $S_\infty(J, s)$ executes each job as soon as it becomes eligible to execute, thereby leaving as little work to be done later as possible, every schedule for J on speed-s processors meeting all deadlines has to complete at least $\mathsf{work}(J, I, s)$ units of execution over interval I.
- For any positive integer t, let $\mathsf{work}(J, t, s)$ denote the maximum value $\mathsf{work}(J, I, s)$ can take, over any interval I of duration equal to t.
- Finally, for any sporadic real-time SDFG G, let $\mathsf{work}(G, t, s)$ denote the maximum value of $\mathsf{work}(J, t, s)$, over all job sequences J corresponding to actor firings that could be required for the correct execution of G.

Above we have defined $\mathsf{work}(G, t, s)$ to be the *maximum* value of $\mathsf{work}(J, t, s)$, over all job sequences J that correspond to actor firings that could be required for the correct execution of G. It is evident that this maximum is achieved when the deadline of some iteration of G coincides with the rightmost endpoint of an interval of duration t, and the other iterations of G occur as closely as legal — i.e., separated by a duration exactly equal to the period T of the SDFG.

Example 4. In Figure 5, we illustrate how the work function is computed by plotting $\mathsf{work}(G, t, 0.5)$ for $0 \leq t \leq 16$, for the example sporadic SDFG of Figure 2. The schedule $S_\infty(J, 0.5)$ for this task with J representing the jobs generated when external tokens arrive at time-instants 0 and $T = 10$ is depicted in Figure 4. Let us consider the deadline marked by an inverted arrow at time-instant 16 (in red, for those reading upon a color medium); observe that the two iterations of the SDFG depicted in this figure occur exactly a period apart. Below we examine the intervals $[16 - t, 16]$ in $S_\infty(J, 0.5)$ for different values of t.

- $\mathbf{t \leq 2}$. Although (jobs corresponding to) actors a and b execute over the interval $[14, 16]$ in $S_\infty(J, 0.5)$, both these jobs have deadlines beyond time-instant 16.
- $\mathbf{2 \leq t \leq 6}$. Both jobs executing over the interval $[10, 14]$ in $S_\infty(J, 0.5)$ have deadlines prior to 16, and hence "contribute" to the work function. $\mathsf{work}(G, t, 0.5)$ therefore increases with a slope $s = 0.5$.
- $\mathbf{6 \leq t \leq 8}$. Since no execution occurs in $S_\infty(J, 0.5)$ over the interval $[8, 10]$, the work function remains unchanged.
- $\mathbf{8 \leq t \leq 10}$. Although actor b executed in $S_\infty(J, 0.5)$ over the interval $[6, 8]$, this actor's deadline is at time-instant 36, which is beyond 16. Therefore, the work function remains unchanged.

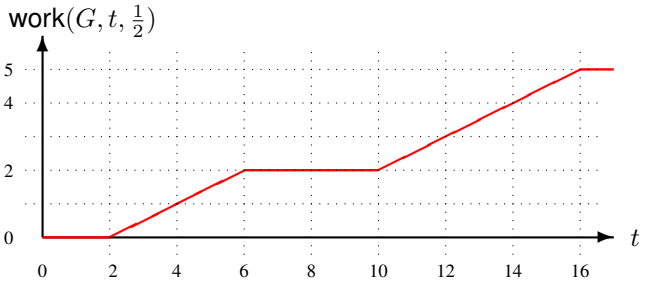


Fig. 5: The work function derived in Example 4.

- $\mathbf{10 \leq t \leq 12}$. Both a and b execute in $S_\infty(J, 0.5)$ during $[4, 6]$. While actor a's deadline is at 16, actor b's deadline is > 16; hence only actor a's execution contributes to the work function.
- $\mathbf{12 \leq t \leq 16}$. Both actors that execute in $S_\infty(J, 0.5)$ during $[0, 4]$ have their deadlines ≤ 16, and therefore contribute to the work function.

Putting the pieces together and simplifying, we have the following expression for $\mathsf{work}(G, t, 0.5)$ for $0 \leq t \leq 16$ (also depicted visually in Figure 5):

$$\mathsf{work}(G, t, 0.5) = \begin{cases} 0 & t \leq 2 \\ 0.5 \times (t-2) & 2 \leq t \leq 6 \\ 2 & 6 \leq t \leq 10 \\ 2 + 0.5 \times (t-2) & 10 \leq t \leq 16 \end{cases} \quad (6)$$

Continuing our analysis of this example, we saw that since $\mathbf{s}(a)$ and $\mathbf{s}(b)$ are both > 0 neither contributes to $\mathsf{work}(G, t, 0.5)$ for values of $t < D$ (which equals 6). We also saw that, since $\mathbf{s}(a) = 1$, the execution of actor a contributes to $\mathsf{work}(G, t, 0.5)$ for values of $t \geq 6$ — in Figure 5, it is actor a's execution that results in $\mathsf{work}(G, t, 0.5)$ increasing at a rate $\frac{1}{2}$ for $t \in [10, 12]$. In a similar vein, it turns out that since $\mathbf{s}(b) = 3$, the execution of actor b only contributes to $\mathsf{work}(G, t, 0.5)$ for values of $t \geq 2T + D$ which equals $2 \times 10 + 6$ or 26; this is depicted in the extended version of the plot of Figure 5 above, which may be found as Figure 6. □

Generalizing from the example above, it can be shown that each actor v of a sporadic real-time HSDFG G contributes to $\mathsf{work}(G, t, s)$ for values of $t \geq \big((\mathbf{s}(v) - 1) \times T + D\big)$, where $\mathbf{s}(v)$ denotes the computed skip vector value for actor v and D and T denote the relative deadline and period of G. Let $\lambda_1, \lambda_2, \ldots, \lambda_k$ denote the distinct non-zero values of $\mathbf{s}(v)$ for actors v of G, sorted in increasing order (i.e., $\lambda_i < \lambda_{i+1}$ for all i). It is fairly straightforward to establish that

- As a function of t, the work function $\mathsf{work}(G, t, s)$ increases in a *periodic* manner (equivalently, its time-derivative is periodic) with a period T, for values of t in (i) $[D, (\lambda_1 - 1)T + D]$; (ii) $[(\lambda_\ell - 1)T + D, (\lambda_{\ell+1} - 1)T + D]$, for each $\ell, 1 \leq \ell < k$; and (iii) $[(\lambda_k - 1)T + D, \infty]$.
- $\mathsf{work}(G, t, s)$ is a piecewise linear function, and the number of linear "pieces" within any interval of duration T is bounded from above by the number of actors in G.

It immediately follows that for any sporadic real-time HSDFG G, an expression like Expression 6 for $\mathsf{work}(G, t, s)$ can be determined in time polynomial in the representation of G.

Example 5. For the example HSDFG G of Figure 2, we have $\lambda_1 = 1$, $(\mathbf{s}(a))$ and $\lambda_2 = 3$ $(\mathbf{s}(b))$. Figure 6 depicts the work function for this HSDFG over $[0, 50]$; it may be verified that

- The work function trivially increases periodically with period $T = 10$ over the interval
$$[D, (\lambda_1 - 1)T + D] = [6, 0 \times 10 + 6] = [6, 6],$$
since the duration of the interval is ≤ 10;
- The work function increases periodically with period 10 over the interval
$$[(\lambda_1 - 1)T + D, (\lambda_2 - 1)T + D] = [6, (3-1) \times 10 + 6] = [6, 26],$$
remaining "flat" for four time units and increasing with slope 0.5 for the remaining six; and
- The work function increases periodically with period 10 over the interval
$$[(\lambda_2 - 1)T + D, \infty] = [(3-1) \times 10 + 6, \infty] = [26, \infty],$$
where it is flat for 2 time units, increases with slope 0.5 for two time units, slope 1 for two more time units, and again with slope 0.5 for a further four time units.

B. A schedulability test

Let τ denote a system of multiple independent sporadic real-time HSDFGs, and let us extend the definition of the work function from individual sporadic real-time SDFGs to τ in the obvious manner: $\mathsf{work}(\tau, t, s) = \sum_{G \in \tau} \mathsf{work}(G, t, s)$.

We have established, in Lemma 4, that the jobs generated by a system of multiple independent sporadic real-time HSDFGs τ in response to legal sequences of arrivals of external tokens constitute a well-formed collection of jobs. By applying Lemmas 1 and 3, we can therefore conclude that the following together constitute sufficient conditions for the jobs generated by such a system τ in response to any legal sequence of arrivals of external tokens to be correctly scheduled by the EDF-based algorithm of [19] upon m speed-α processors:

1) The sum of the WCETs of each sequence of precedence-constrained jobs generated by each HSDFG in τ not exceed the deadline of the job at the end of the sequence relative to the release of the first job of the sequence, and
2) There is a schedule meeting all deadlines for these jobs that completes no more than $(\alpha m - m + 1)|I|$ units of execution within any interval I.

As stated right after Lemma 3, the first of these conditions can be verified for each HSDFG in the system, in time pseudo-polynomial in its representation. Observe that for the second condition to hold, it is sufficient that $\mathsf{work}(\tau, |I|, 1) \leq (\alpha m - (m-1)) \times |I|$ for all intervals I. We therefore conclude that

Lemma 5. *Let α denote any constant ≥ 1. A system of multiple independent sporadic real-time HSDFGs τ is schedulable on m speed-α processors by the EDF-based scheduling*

algorithm of [19] if each HSDFG in the system satisfies the condition of Lemma 3 upon unit-speed processors, and

$$\mathsf{work}(\tau, t, 1) \leq (\alpha m - (m-1)) \times t$$

for all values of $t \geq 0$.

In performing schedulability analysis, it is typical to assume that we have *unit-speed* processors available to us, and to determine whether a given system is guaranteed to be scheduled to meet all deadlines upon a platform comprising such processors. We now restate the result of Lemma 5 in a manner that allows us to answer this question. Let σ denote any constant < 1. It is straightforward to mimic the derivation of Lemma 1 with "unit-speed" replaced by σ and α replaced by 1, to get

Theorem 2. *Let* σ *denote any constant* < 1. *A system of multiple independent sporadic real-time SDFGs* τ *is schedulable on* m *unit-speed processors by the EDF-based scheduling algorithm of [19] if each HSDFG in the system satisfies the condition of Lemma 3 upon speed-*σ *processors, and*

$$\mathsf{work}(\tau, t, \sigma) \leq (m - (m-1)\sigma) \times t \tag{7}$$

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 2 above, to first verify that each HSDFG in the system satisfies the condition of Lemma 3 upon speed-σ processors and if so, to produce a value for σ such that Condition 7 holds for all $t \geq 0$. We refer to such a σ as a *witness* to the EDF-schedulability of τ. To show that a given system of sporadic real-time HSDFGs is EDF schedulable, we need to produce a witness to its schedulability. A sufficient schedulability test with speedup $(2 - 1/m)$ is therefore immediately obtained by checking whether $\sigma \leftarrow m/(2m-1)$ is a witness, and declaring the task system EDF-schedulable if the answer is "yes." That is, the following constitutes ***a speedup-optimal sufficient schedulability test*** for the EDF-based scheduling algorithm of [19]:

1) Each HSDFG in τ should satisfy the condition of Lemma 3 upon speed-$m/(2m-1)$ processors, and
2) For all values of $t \geq 0$,

$$\mathsf{work}\big(\tau, t, \frac{m}{2m-1}\big) \leq \big(m - \frac{m(m-1)}{2m-1}\big) \times t \tag{8}$$

Let us define the ***normalized utilization*** $\mathcal{U}(\tau)$ of a system of sporadic real-time SDFGs τ that is to be implemented upon m unit-speed processors as follows:

$$\mathcal{U}(\tau) \stackrel{\text{def}}{=} \Big(\sum_{G \in \tau} U(G)\Big)/m,$$

where $U(G)$ is as defined in Expression 2. Using arguments essentially identical to those used in multiprocessor EDF schedulability analysis for sporadic DAG tasks (or even for simpler 3-parameter sporadic tasks), it is straightforward to show that for task systems with normalized utilization bounded from above by a constant strictly less than one, Condition 8 needs to be checked in at most pseudo-polynomially many distinct values of t. This immediately yields the following result:

Theorem 3. *Speedup-optimal sufficient EDF-based schedulability analysis for systems of sporadic real-time HSDFGs can be done in pseudo-polynomial time, for systems with normalized utilization bounded from above by a constant strictly less than one.*

Extending Theorem 3 to general SDFGs. Theorem 3 establishes that schedulability analysis for systems of homogeneous SDFGs can be done in pseudo-polynomial time. What about general (i.e., not homogeneous) SDFGs? Here our suggested approach is to first convert each SDFG to an equivalent HSDFG using the algorithm of [12], and then apply the schedulability test of this section. Since the conversion algorithm of [12] may take exponential time and yield an HSDFG of size exponential in the size of the original SDFG, this implies that our schedulability analysis test may also take time exponential in the representation of the SDFG (although pseudo-polynomial in the representation of the equivalent HSDFG). Perhaps this is not particularly unreasonable: it has been observed that the conversion algorithm of [12] takes exponential time and yields an HSDFG of size exponential in the size of the original SDFG only in pathological cases, when there is some inherent exponential blowup in the run-time "behavior" of the SDFG. If this is the case and the behavior of an SDFG is in fact exponentially-sized, perhaps it is overly ambitious to be able to validate schedulability properties in less than exponential time; we pose further consideration of this question as an interesting open problem.

VI. Summary

The Synchronous Data Flow Graph (SDFG) model is widely used in the modeling of embedded real-time systems. In this paper we have made an effort to apply real-time scheduling theory to obtain a better understanding of the problem of achieving highly resource-efficient implementations of SDFG-modeled real-time systems. The first optimal algorithm for dynamically scheduling a collection of such tasks upon a preemptive uniprocessor platform has recently been proposed [19], [20]; here, we have extended this algorithm, and its analysis, to identical multiprocessor platforms. We have proved that the algorithm of [19] is speedup-optimal amongst all EDF-based algorithms for solving this problem, and have designed a speedup-optimal sufficient schedulability-analysis test.

Acknowledgements

This research has been supported in parts by NSF grants CNS 1409175 and CPS 1446631, AFOSR grant FA9550-14-1-0161, and ARO grant W911NF-14-1-0499.

References

[1] H. I. Ali, B. Akesson, and L. M. Pinho, "Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs," in *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, March 2015, pp. 701–710.

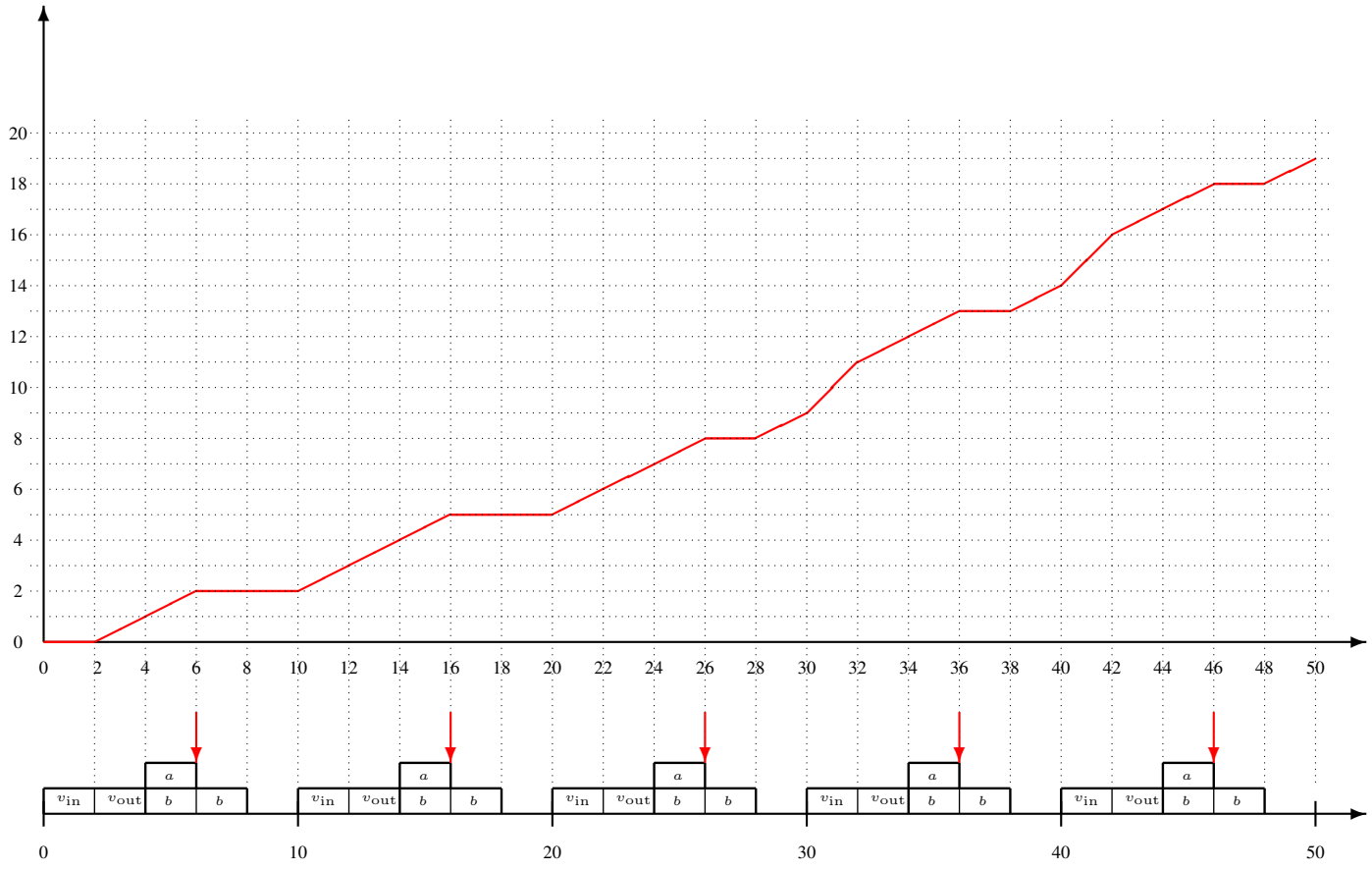


Fig. 6: The work function of Example 4, extended further out along the time-line than in Figure 5.

[2] M. Bamakhrama and T. Stefanov, "Hard-real-time scheduling of data-dependent tasks in embedded streaming applications," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, ser. EMSOFT '11. New York, NY, USA: ACM, 2011, pp. 195–204. [Online]. Available: http://doi.acm.org/10.1145/2038642.2038672

[3] M. A. Bamakhrama and T. Stefanov, "Managing latency in embedded streaming applications under hard-real-time scheduling," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 83–92. [Online]. Available: http://doi.acm.org/10.1145/2380445.2380464

[4] 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.

[5] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*. Springer Publishing Company, Incorporated, 2015.

[6] 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.

[7] 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.

[8] A. Bouakaz, T. Gautier, and J. P. Talpin, "Earliest-deadline first scheduling of multiple independent dataflow graphs," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2014, pp. 1–6.

[9] A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen, "Latency minimization for synchronous data flow graphs," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, Aug 2007, pp. 189–196.

[10] J. Khatib, A. M. Kordon, E. C. Klikpo, and K. Trabelsi-Colibet, "Computing latency of a real-time system modeled by synchronous dataflow graph," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, 2016, pp. 87–96. [Online]. Available: http://doi.acm.org/10.1145/2997465.2997479

[11] E. C. Klikpo and A. M. Kordon, "Preemptive scheduling of dependent periodic tasks modeled by synchronous dataflow graphs," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, 2016, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/2997465.2997474

[12] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 24–35, January 1987.

[13] ——, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[14] E. A. Lee, "A coupled hardware and software architecture for programmable digital signal processors," Ph.D. dissertation, EECS Department, University of California, Berkeley, 1986. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1986/715.html

[15] M. Mohaqeqi, J. Abdullah, and W. Yi, *Modeling and Analysis of Data Flow Graphs Using the Digraph Real-Time Task Model*. Cham: Springer International Publishing, 2016, pp. 15–29. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-39083-3_2

[16] 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.
[17] S. Neuendorffer, "The SDF Domain," in *Heterogeneous Concurrent Modeling and Design in Java*, C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng, Eds. EECS, University of California, Berkeley, 7 2005, vol. 3 (Ptolemy II Domains), ch. 3, pp. 49–60, memorandum UCB/ERL M05/23.
[18] C. A. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal time-critical scheduling via resource augmentation," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, El Paso, Texas, 4–6 May 1997, pp. 140–149.
[19] A. Singh, P. Ekberg, and S. Baruah, "Applying real-time scheduling theory to the synchronous data flow model of computation," in *2017 29th Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, June 2017.
[20] ——, "Uniprocessor scheduling of real-time synchronous dataflow tasks," 2017, Under review.
[21] F. Siyoum, "Worst-case temporal analysis of real-time dynamic streaming applications," Ph.D. dissertation, PhD thesis, Eindhoven University of Technology, 2014.

Appendix: Proof of Lemma 1

Let J denote an arbitrary collection of well-formed jobs, for which each arrival-time, execution time, and deadline are integers. Let $S_\infty(J)$ denote a work-conserving schedule for J upon an unbounded number of unit-speed processors. We note two properties of $S_\infty(J)$:

Property (K1): Each job begins and completes execution at an integer time-instant.

Property (K2): $S_\infty(J)$ *dominates* all valid schedules for J upon unit-speed processors: at any point in time and for any job, $S_\infty(J)$ has executed at least as much of that job as any valid schedule for J upon unit-speed processors.

Definition 4. *Let an (m,α)-**counterexample** denote a well-formed collection of jobs J that is scheduled correctly in $S_\infty(J)$, but not by EDF on m speed-α processors ($m \in \mathbb{N}_{\geq 2}, \alpha > 1$).*

*A **minimal** (m,α)-**counterexample** is an (m,α)-counterexample J, for which each subset $J' \subsetneq J$ is scheduled correctly by EDF on m speed-α processors.* □

We will use the lemma below to prove the correctness of Lemma 1.

Lemma 6. *If J is a minimal (m,α)-counterexample then there is some interval I such that any valid schedule for I must complete strictly more than $\big(\alpha m - (m-1)\big) \cdot |I|$ units of execution within the interval I.*

Proof: Let t_f denote the earliest time-instant at which EDF[4] misses a deadline when scheduling J. Let t^* denote the latest instant at which EDF has executed at least as much of each job as $S_\infty(J)$ has. (We note that t^* is well-defined since at time zero EDF has completed at least as much of each job as $S_\infty(J)$ has.) Let $I \stackrel{\text{def}}{=} [t^*, t_f]$. Let X denote the total length of all the intervals in I during which all m processors are busy in the EDF schedule. Let $Y \stackrel{\text{def}}{=} |I| - X$, and let $Y_1, Y_2, \ldots,$ denote the (distinct non-contiguous) intervals during which at least one processor is idled in this EDF schedule.

Observe that for any $t > t^*$,

$$\left|[t^*, t] \cap \bigcup_\ell Y_\ell\right|$$

denotes the cumulative duration of the intervals in $[t^*, t]$ during which at least one processor is idled in the EDF schedule.

Let us define t_o to be the smallest t such that

$$\alpha \times \left|[t^*, t] \cap \bigcup_\ell Y_\ell\right| \geq \big(\lceil t^* \rceil - t^*\big)$$

Since at least one processor is idled at each instant during the interval $\left|[t^*, t_o] \cap \bigcup_\ell Y_\ell\right|$, and this entire interval lies after t^* (i.e., to the right of t^* on the time-line), any job executing in $S_\infty(J)$ during $[t^*, \lceil t^* \rceil]$ will either have already completed, or be executing in EDF during this interval. Therefore, EDF has executed at least as much of each job by t_o as $S_\infty(J)$ has by $\lceil t^* \rceil$.

Observe that $t_o \geq \lceil t^* \rceil$ – otherwise, t_o would be a *later* instant than t^* at which EDF has completed at least as much of each job as $S_\infty(J)$ – a contradiction.

Define t_i for $i = 1, 2, \ldots, (t_f - \lceil t^* \rceil)$ in a manner similar to how t_o was defined: t_i is the smallest t such that

$$\alpha \left|[t^*, t] \cap \bigcup_\ell Y_\ell\right| \geq (\lceil t^* \rceil - t^*) + i \tag{9}$$

Claim 1. *For each $i \geq 0$*

1) *EDF has executed at least as much of each job by time t_i as $S_\infty(J)$ by time $\lceil t^* \rceil + i$, and*
2) *$t_i \geq \lceil t^* \rceil + i$.*

Proof of Claim 1: By induction on i.
BASE CASE ($i = 0$). Explicitly shown above.
INDUCTIVE STEP. Assume, as the inductive hypothesis (IH), that it's true for i; we will establish it for $i+1$.
To prove the first claim:

- By Property (K1) and the fact, following from the IH, that $t_i \geq (\lceil t^* \rceil + i)$, it must be the case that all jobs that were executed in $S_\infty(J)$ during $[\lceil t^* \rceil + i, \lceil t^* \rceil + (i+1)]$ are available for EDF at each time-instant in the interval $\Big([t_i, t_{i+1}] \cap \bigcup_\ell Y_\ell\Big)$ (or have already completed in the EDF schedule).
- Since there is at least one processor idled by EDF throughout $\Big([t_i, t_{i+1}] \cap \bigcup_\ell Y_\ell\Big)$, it must be the case that all jobs executed in $S_\infty(J)$ during $[t^* + i, t^* + (i+1)]$ that have *not* completed in EDF will be executed in the EDF schedule during $\Big([t_i, t_{i+1}] \cap \bigcup_\ell Y_\ell\Big)$.

To prove the second claim, we observe that otherwise t_{i+1} would be a *later* instant than t^* at which EDF has completed at least as much of each job as $S_\infty(J)$ – this contradicts the definition of t^* as the latest instant at which this happens.

This completes the proof of Claim 1. □

[4]Throughout this proof, the term "EDF" should be taken to mean "EDF executing upon m speed-α processors".

Let $\hat{i} \stackrel{\text{def}}{=} t_f - \lceil t^* \rceil$. By Claim 1 above, EDF will have executed at least as much of each job by time-instant $t_{\hat{i}}$ as $S_\infty(J)$ does by t_f. But all jobs are assumed to meet their deadlines in $S_\infty(J)$; hence, for EDF to fail to schedule J correctly it is necessary that $t_{\hat{i}} > t_f$. Equivalently, it must be the case that even if the entire range $[t^*, t_f]$ were considered in the LHS of Equation 9, we would not be able to define $t_{\hat{i}}$:

$$\begin{aligned} & \alpha \Big| [t^*, t_f] \cap \bigcup_\ell Y_\ell \Big| < (\lceil t^* \rceil - t^*) + \hat{i} \\ \Leftrightarrow \quad & \alpha Y < (\lceil t^* \rceil - t^*) + \hat{i} \\ \Leftrightarrow \quad & \alpha Y < \big(t_f - t^*\big) \\ \Leftrightarrow \quad & \alpha Y < |I| \end{aligned}$$

Therefore during the interval I EDF completes at least

$$\begin{aligned} \alpha(mX + Y) &= \alpha(m(|I| - Y) + Y) \\ &= \alpha(m|I| - (m-1)Y) \\ &= \alpha m|I| - (m-1)\alpha Y \\ &> \alpha m|I| - (m-1)|I| \\ &= \big(\alpha m - (m-1)\big)|I| \end{aligned}$$

This completes the proof of Lemma 6. □

We are now ready to prove the correctness of Lemma 1 (replicated below):

Lemma 1. *Consider a well-formed collection of jobs J. Let $m \in \mathbb{N}$ denote any positive integer, and $\alpha \in \mathbb{R}_{\geq 1}$ any constant that is ≥ 1.* ***If*** *there exists some schedule meeting all deadlines for J upon unit-speed processors which, for any time interval I, completes no more than $(\alpha m - m + 1) \cdot |I|$ units of execution within I,* ***then*** *J is global-EDF schedulable on m speed-α processors.*

Let J denote a well-formed collection of jobs for which there exists a schedule upon unit-speed processors meeting all deadlines. Since such a schedule exists, if follows that $S_\infty(S)$ meets all deadlines. If J were not global EDF schedulable upon m unit-speed processors, then there is some $J' \subset J$ that constitutes a minimal (m, α)-counterexample. By Lemma 6 above, this would require that there exist some interval I over which any valid schedule for J' (and therefore, any valid schedule for J) completes $> (\alpha m - m + 1) \cdot |I|$ units of execution within I, thereby contradicting the statement of Lemma 1. □