A Soft-Real-Time Optimal Scheduler for DAG Tasks with Node-Level Self Dependencies

Shareef Ahmed[‡†] and James H. Anderson[†]
†*University of North Carolina at Chapel Hill, USA*
‡*University of South Florida, USA*
Email: shareefahmed@usf.edu, anderson@cs.unc.edu

***Abstract*—Modern real-time workloads are often expressed as processing graphs that have complex dataflow dependencies. No scheduling algorithm with a holistic analysis of graph-based tasks is known that can provide bounded response times without utilization loss, thus ensuring soft-real-time optimality, when a node instance depends on some of its prior instances and multiple invocations of the same graph can be active simultaneously. This paper presents a scheduling policy for such a graph-based task and provides a response-time analysis that guarantees bounded response times without utilization loss. Experimental evaluations show that our scheduler yields significantly tighter response-time bounds than existing soft-real-time optimal schedulers.**

I. Introduction

Many modern real-time systems can be expressed as *directed acyclic graphs* (DAGs) formed by dataflow dependencies among different tasks. Such DAG-based tasks are usually computationally demanding, necessitating the use of multiple processing cores for timely completion. Scheduling DAG-based tasks on multicore platforms has been a major focus of real-time systems research over the last decade [1], [4], [16], [29], [37], [38], [41], [42], [44].

Unfortunately, scheduling and response-time analysis of DAG tasks become complicated when multiple instances of the same DAG task can be present at the same time. Such scenarios occur in hard real-time (HRT) systems with task deadlines exceeding task periods and in soft real-time (SRT) systems where only bounded response times are required. Practical use cases exist where such functionality is needed. For example, in autonomous vehicles, where processing graphs are commonly used, it may be necessary to overlap the processing of successive video frames to provide acceptable throughput. Further complications arise when the jobs (instances) of a node in these DAG tasks have *self-dependencies* (due to data dependencies), likely needing information from the processing of a prior video frame in an object-tracking application, meaning that a job of a node cannot commence execution until a certain past job of the same node completes. Consequently, with a few exceptions [1], [4], [25], [26], most work on DAG scheduling either precludes node-level self-dependencies or allows only one instance of a DAG at a time (thus, satisfying node-level self-dependencies implicitly).

Work supported by NSF grants CPS 2038960, CPS 2038855, CNS 2151829, and CPS 2333120, and a faculty startup fund from the University of South Florida.

Analyzing DAGs with node-level self-dependencies. Most DAG scheduling approaches that allow node-level self-dependencies adopt a *decomposition*-based approach where a graph is decomposed into subDAGs that are independently analyzed. Such a decomposition is typically performed at the node level: a response-time bound of each node is determined independently and the maximum among summed bounds along different paths of the DAG represents a DAG response-time bound. Remarkably, such approaches achieve *soft-real-time optimality* by ensuring bounded response times without causing any *utilization loss* under certain global scheduling techniques [4], [25]. Unfortunately, each node's response-time bound can be pessimistically large, causing a large and loose DAG response-time bound.

To ameliorate such pessimism, a non-decomposition-based response-time analysis technique is needed. Such approaches usually determine a DAG's response-time bound by examining its schedule along a certain path (*e.g.*, [12], [16], [39]). However, to our knowledge, *no scheduler is known that allows non-decomposition-based analysis of even a single DAG task with node-level self-dependencies and ensures bounded response times without sacrificing any processing capacity.*[1]

In this paper, we present a soft-real-time optimal scheduler for a DAG task with node-level self-dependencies under a model called the *restricted-parallelism* (*rp*) model. Under the rp model, each node of a DAG has a *degree of parallelism* that defines how many consecutive jobs of the node can execute concurrently. (In Sec. II, we discuss how this term also represents a node's degree of self-dependency.) Our scheduling algorithm is based on the principle that when multiple DAG instances are present at the same time, each of them must progress toward completion concurrently to ensure a bounded response time for each instance. (We show the necessity of this principle in Sec. IV.) To ensure such progression, our scheduling algorithm may increase (*boost*) the priority of a job at most once during the job's execution. Such priority changes are commonly used in real-time systems to ensure that a job accessing a mutually exclusive shared resource progresses towards completion despite having a low priority. We give

[1]For any constrained-deadline DAG task, processing capacity loss can be unavoidable, as some processors may be idle due to a lack of ready jobs. However, for SRT DAG tasks or HRT DAG tasks with sufficiently large deadlines, all processors can be busy by having enough ready jobs from different DAG invocations in the long run.

a response-time analysis technique that can utilize recently proposed *multi-path* bounds for a constrained-deadline DAG task [16], [39].

Contributions. Our contribution is threefold. First, we propose a scheduling algorithm for an rp-sporadic DAG task. Our scheduling algorithm works even when different nodes of a DAG have different degrees of parallelism. Second, we provide a *coarse-grained* (looser) and a *fine-grained* (tighter) response-time bound for a DAG task scheduled under this approach, provided that the underlying platform is not over-utilized. Finally, we present an experimental evaluation that illustrates the benefits of our scheduling algorithm.

Organization. After covering needed background (Sec. II), we review multi-path bounds for a non-recurrent DAG task (Sec. III), describe challenges in scheduling a DAG task with node-level self-dependencies (Sec. IV), give our scheduling algorithm for a DAG task and derive its response-time bounds (Secs. V and VI), discuss our experiments (Sec. VII), review relevant prior work (Sec. VIII), and conclude (Sec. IX).

II. Background

In this section, we provide needed definitions; Tbl. I summarizes the notation given here.

Task model. We consider a DAG task G to be scheduled on m unit-speed processors. DAG task G is represented as a tuple (V, E), where V and E are sets of nodes and directed edges, respectively. V consists of n nodes that represent sequential tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. The *worst-case execution time* (WCET) of τ_i is denoted by C_i. Without loss of generality, we assume the following.

I. Tasks of G are indexed according to a topological ordering of G.

A directed edge from τ_i to τ_k represents a precedence constraint between the *predecessor* task τ_i and the *successor* task τ_k. The set of predecessors (resp., successors) of τ_i is denoted by $pred(\tau_i)$ (resp., $succ(\tau_i)$). We assume that each DAG task G has a unique *source* task τ_1 with no incoming edges and a unique *sink* task τ_n with no outgoing edges.[2]

A *path* $\lambda = \langle v_1, v_2, \ldots, v_k \rangle$ is a sequence of nodes of G (*i.e.*, $v_i \in V$ for each $1 \leq i \leq k$) such that $v_i = pred(v_{i+1})$ holds. (We use the symbol v, instead of τ, to simplify the indexing of nodes in λ.) If a path exists from τ_i to τ_k, then τ_i (resp., τ_k) is called an *ancestor* (resp., *descendant*) of τ_k (resp., τ_i). The set of ancestors of τ_i is denoted as $anc(\tau_i)$. We define the *length* of a path and the *volume* of a set of tasks as follows.

Def. 1. *The* length $len(\lambda)$ *of a path* λ *is the sum of the WCETs of the nodes on* λ*, i.e.,* $len(\lambda) = \sum_{\tau_i \in \lambda} C_i$*. We denote the length of the longest path from the source task* τ_1 *to the sink task* τ_n *by* $len(G)$*.*

[2] A DAG with multiple sources/sinks can be supported by adding a "virtual" source or sink with a WCET of zero.

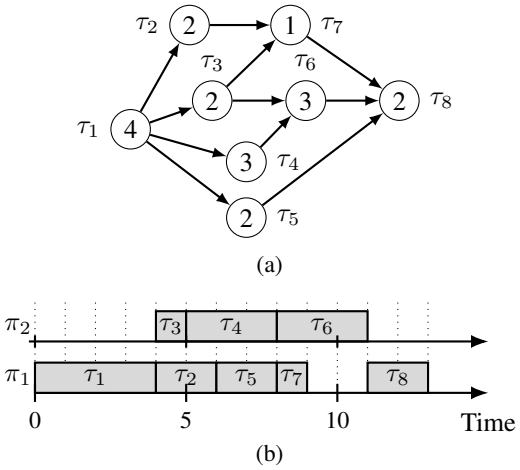


Fig. 1: (a) A DAG G (numbers inside circles denote WCETs). (b) A schedule of G where τ_3 executes for less than its WCET.

Def. 2. *The* volume *of any subset of tasks* $V' \subseteq V$ *is* $vol(V') = \sum_{\tau_i \in V'} C_i$*. The volume of* G *is* $vol(G) = \sum_{i=1}^{n} C_i$*.*

Ex. 1. Fig. 1(a) depicts a DAG task G. The sequence $\lambda = \langle \tau_1, \tau_4, \tau_6, \tau_8 \rangle$ is a path with $len(\lambda) = 12$. Path λ is also the longest path from τ_1 to τ_8. Thus, $len(G) = 12$. The volume of G is $vol(G) = 19$. Task τ_6's ancestors (resp., descendants) are $anc(\tau_6) = \{\tau_1, \tau_3, \tau_4\}$ (resp., $desc(\tau_6) = \{\tau_8\}$). $\Diamond$

Each DAG task G releases a potentially infinite sequence of *DAG jobs* $J_1, J_2, \ldots$. These DAG jobs are released at least T time units apart, where T is the *period* of G. The *release time* and *finish time* of DAG job J_j are denoted by $r(J_j)$ and $f(J_j)$, respectively. The DAG job J_j consists of a *job* $\tau_{i,j}$ for each task τ_i in that DAG. The *release time* and *finish time* of $\tau_{i,j}$ are denoted by $r(\tau_{i,j})$ and $f(\tau_{i,j})$, respectively. The source task τ_1 releases its j^{th} job when J_j is released, *i.e.*, $r(J_j) = r(\tau_{1,j})$. The j^{th} job of each non-source task is released once the j^{th} job of each of its predecessors finishes, *i.e.*, $r(\tau_{i,j}) = \max_{\tau_k \in pred(\tau_i)}\{f(\tau_{k,j})\}$. DAG job J_j finishes execution when $\tau_{n,j}$ finishes, *i.e.*, $f(J_j) = f(\tau_{n,j})$. The *response time* of $\tau_{i,j}$ is $f(\tau_{i,j}) - r(\tau_{1,j})$. Task τ_i's response time is $R(\tau_i) = \max_j\{f(\tau_{i,j}) - r(\tau_{1,j})\}$. J_j's response time equals $\tau_{n,j}$'s response time. G's response time equals τ_n's response time, *i.e.*, $R(\tau_n)$.

DAG G has a *relative deadline* D. DAG job J_j's *deadline* is $d(J_j) = r(J_j) + D$. We consider *soft* deadlines, *i.e.*, a DAG job can miss its deadline; however, we require each DAG job to have a bounded response time. The *utilization* of τ_i is $u_i = C_i/T$. The utilization of G is $U = \sum_{i=1}^{n} u_i = vol(G)/T$. We consider preemptive scheduling.

Ex. 1 (Cont'd). Fig. 1(b) shows a schedule of a DAG job J_1 of G on two processors π_1 and π_2. (Job indices are omitted in Fig. 1(b).) J_1 (hence, $\tau_{1,1}$) is released at time 0. Jobs of τ_2, τ_3, τ_4, and τ_5 are released when the job of τ_1 completes at time 4. Note that τ_3 executes for less than its WCET. $\Diamond$

Restricted parallelism. We specify node-level self-

TABLE I: Notation summary.

Symbol	Meaning	Symbol	Meaning
G	A DAG task	D	G's relative deadline
m	No. of processors	λ	A path of G
n	No. of nodes	$len(\cdot)$	Def. 1
τ_i	i^{th} task of G	P_i	τ_i's degree of parallelism
T	Period of G	$anc(\tau_i)$	τ_i's ancestor
C_i	WCET of τ_i	$desc(\tau_i)$	τ_i's descendants
$\tau_{i,j}$	j^{th} job of τ_i	$r(\cdot)$	Release time
$pred(\tau_i)$	τ_i's predecessors	$f(\cdot)$	Finish time
$succ(\tau_i)$	τ_i's successors	$R(\cdot)$	Response time
$vol(\cdot)$	Def. 2	$C_{i,j}(\cdot)$	Def. 6
u_i	C_i/T	V_k	Def. 7
J_j	j^{th} DAG job	$G(\cdot)$	Def. 8
U	$\sum_{i=1}^{n} u_i$	$d(J_j)$	Deadline of J_j

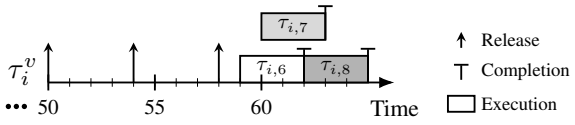


Fig. 2: Example illustrating restricted parallelism. Subsequent jobs are shaded darker.

dependencies according to a previously introduced model called the *restricted-parallelism (rp)* model [4]. Under the rp model, each task τ_i has a parameter P_i, called its *degree of parallelism*, that defines self-dependencies among τ_i's jobs as follows: job $\tau_{i,j}$ with $j > P_i$ cannot start execution until $\tau_{i,j-P_i}$ completes. Note that P_i also represents the number of consecutive jobs of τ_i that can execute in parallel, *e.g.*, job $\tau_{i,j}$ can execute in parallel with jobs $\tau_{i,j-P_i+1}, \tau_{i,j-P_i+2}, \ldots, \tau_{i,j-1}$. Under the rp model, a job's readiness is defined as follows.

Def. 3. *A DAG job J_j is* pending *at time t if and only if $r(J_j) \leq t < f(J_j)$ holds. A job $\tau_{i,j}$ is* pending *at time t if and only if $r(J_j) \leq t < f(\tau_{i,j})$ holds. A job $\tau_{i,j}$ is* ready *at time t if and only if $r(\tau_{i,j}) \leq t < f(\tau_{i,j})$ and $f(\tau_{i,j-P_i}) \leq t$ (if $j > P_i$) hold.*

Note that, by the definition of $r(\tau_{i,j})$, the j^{th} jobs of all predecessor nodes of τ_i have finished execution by time $r(\tau_{i,j})$. Under the rp model, τ_i has *no* parallelism (*immediate* self-dependency) if $P_i = 1$. In contrast, τ_i has *unrestricted* parallelism (*no* self-dependency) if $P_i = \infty$.

Ex. 2. Fig. 2 depicts a task τ_i with $P_i = 2$. Assume that τ_i is scheduled on three processors. Jobs $\tau_{i,6}, \tau_{i,7}$, and $\tau_{i,8}$ are released at times 50, 54, and 58, respectively. At time 59, $\tau_{i,6}$ is scheduled. Assume that $\tau_{i,7}$ and $\tau_{i,8}$ are among the m highest-priority jobs that are released but have not completed execution at time 60. Since $P_i = 2$, $\tau_{i,7}$ is scheduled at time 60. However, $\tau_{i,8}$ is not scheduled until time 62 when $\tau_{i,6}$ completes execution. ◊

SRT-optimality. A system is *SRT-schedulable* under a scheduling algorithm if each task can be guaranteed to have a bounded response time under the scheduling algorithm. A system is *SRT-feasible* if the system is SRT-schedulable under some scheduling algorithm. A scheduling algorithm is *SRT-optimal* if every SRT-feasible system is SRT-schedulable under the algorithm.

SRT-feasibility conditions. Under the rp model, the following condition ensures that a DAG task G can be scheduled with a bounded response time on m processors [4]:[3]

$$U \leq m \wedge (\forall \tau_i \in V : u_i \leq P_i). \tag{1}$$

Thus, an SRT-optimal scheduler for a DAG task can schedule any DAG task that satisfies (1) with a bounded response time. We assume that G satisfies (1).

Concrete instantiation. In a *concrete instantiation* of a G, the release time of each DAG job and the actual execution time of every job of each node are known. For G, infinitely many concrete instantiations exist.

III. Response-Time Bound for Non-Recurrent DAGs

In this section, we review a recently proposed response-time bound for a DAG task G under any work-conserving preemptive scheduler [16]. The bound is applicable for a non-recurrent DAG task (*i.e.*, a DAG task that releases only one DAG job) or a constrained-deadline DAG task. Since job indices are irrelevant, we omit them in this section. The bound in [16] improves upon the well-known bound by Graham [13]. We begin by illustrating Graham's bound.

Theorem 1 ([13])**.** *The response time R of a DAG G scheduled on m processors under a work-conserving scheduler is bounded as follows.*

$$R \leq len(G) + \frac{vol(G) - len(G)}{m} \tag{2}$$

Theorem 1 relies on the concept of an *envelope path*.

Def. 4. *In a schedule of a concrete instantiation of G, a path $\lambda = \langle v_1, v_2, \cdots, v_k \rangle$ of G is an* envelope path *if and only if*
(i) $v_1 = \tau_1 \wedge v_k = \tau_n$,
(ii) $\forall\, i \in \{1, 2, \ldots, k-1\} : f(v_i) = r(v_{i+1})$,
(iii) $\forall\, i \in \{1, 2, \ldots, k-1\} : v_i \in pred(v_{i+1})$.

Note that an envelope path of G's schedule may not be the longest path of G. Also, an envelope path may not be unique.

Ex. 3. Fig. 1(b) depicts a schedule of a concrete instantiation of G for which $\lambda = \langle \tau_1, \tau_4, \tau_6, \tau_8 \rangle$ is an envelope path of G. Note that τ_4 is released at time 4 when τ_1 finishes. ◊

The following lemma is instrumental in proving Theorem 1 and many other similar bounds [18]. It is based on the observation that whenever no job of an envelope path is executing, all processors must be busy executing jobs that are not on the envelope path. For example, in Fig. 1(b), no jobs of the envelope path execute during the interval $[4, 5)$, and both processors are busy throughout this time interval.

Lemma 1 ([13])**.** *Let G_c be a concrete instantiation of G and $\mathcal{S}$ be a work-conserving schedule of G_c. Let λ_c be an envelope path of G_c in $\mathcal{S}$. Then, at any time t when jobs in λ_c are not scheduled in $\mathcal{S}$, all available processors are busy in $\mathcal{S}$.*

In [16], an improved bound was proposed that considers up to m *generalized paths* (if that many exist) of a DAG,

[3] In [4], a more general feasibility condition was given that applies for multiple DAG tasks.

instead of considering only the envelope path. The notion of *generalized paths* is defined as follows.

Def. 5 ([16]). *A sequence of nodes $\lambda = \langle v_1, v_2, \cdots, v_k \rangle$ is a* generalized path *if and only if $\forall i : v_i = anc(v_{i+1})$ holds. Similar to the length of a path, the length of a generalized path λ is denoted by $len(\lambda) = \sum_{\tau_i \in \lambda} C_i$. A* generalized path list *is a set $\{\lambda_1, \lambda_2, \cdots, \lambda_p\}$ of node-disjoint generalized paths, i.e., $\lambda_i \cap \lambda_j = \emptyset$ for any $1 \leq i < j \leq p$. For ease of notation, we denote $\{\lambda_1, \lambda_2, \cdots, \lambda_p\}$ by $(\lambda_i)_1^p$.*

Ex. 3 (Cont'd). Consider the DAG G shown in Fig. 1(a). $\lambda_1 = \langle \tau_1, \tau_3, \tau_7 \rangle$ is both a path and a generalized path. $\lambda_2 = \langle \tau_4, \tau_8 \rangle$ is not a path, but a generalized path. $(\lambda_i)_1^2 = \{\lambda_1, \lambda_2\}$ is a generalized path list. ◊

Theorem 2 ([16]). *Assume that a DAG G is scheduled on m processors under a work-conserving scheduler. Given a generalized path list $(\lambda_i)_1^p$ (with $1 \leq p \leq m$) of a DAG G where λ_1 is the longest path of G, the response time R of G is bounded as follows.*

$$R \leq \min_{1 \leq j \leq p} \{len(G) + \frac{vol(G) - \sum_{i=1}^{j} len(\lambda_i)}{m - j + 1}\} \quad (3)$$

Thus, Theorem 2 considers a set of at most m generalized paths and selects the minimum value obtained from (3) using this set. Note that (3) matches (2) for $j = 1$. We will apply Theorem 2 in Sec. VI to derive a response-time bound under our proposed scheduler.

IV. Challenges in Scheduling a DAG Task with Node-Level Self-Dependencies

In this section, we briefly describe some of the difficulties in achieving bounded response times via non-decomposition-based analysis for DAG tasks in the presence of node-level self-dependencies.

Unbounded response times under common scheduling policies. Scheduling policies that aid in the non-decomposition-based analysis of DAG tasks often require jobs of the j^{th} DAG job of a DAG to have higher priority than jobs of its $(j+1)^{st}$ DAG job. Schedulers that satisfy this requirement include *global-earliest-deadline-first* (GEDF) and *first-in-first-out* (FIFO) scheduling where jobs are prioritized according to the corresponding DAG jobs' deadlines and release times, respectively. Unfortunately, as shown next, with node-level self-dependencies even for a single node, a DAG task's response time may become unbounded even if its utilization is greater than but close to 1.0.

Ex. 4. Consider the DAG task G with $n = m + 1$ tasks in Fig. 3(a), which is scheduled on m processors. Assume that G's period is T, and the WCET of τ_1 is T. The WCET of each of the remaining m tasks is one time unit. Assume that τ_1's degree of parallelism is $P_1 = 1$. We do not require any specific values of P_i for the remaining tasks. Fig. 3(b) depicts a schedule of G on m processors where jobs of J_i have higher priorities than jobs of J_{i+1}. Assume that each DAG job J_i is released at time $(i-1)T$ and each job executes for its task's WCET. Since the first jobs of $\tau_2, \ldots, \tau_{m+1}$ have higher priorities than the second job of τ_1, it can only be scheduled after one of the first jobs of $\tau_2, \ldots, \tau_{m+1}$ completes at time $T + 1$. Thus, all jobs of J_1 finish execution at time $T + 1$. Similarly, J_k finishes execution at time $(k-1)T + k$. Thus, the response time of G grows unboundedly. Note that G's utilization is $\frac{T+m}{T} = 1 + \frac{m}{T}$, which is greater than but close to 1.0 for large T. ◊

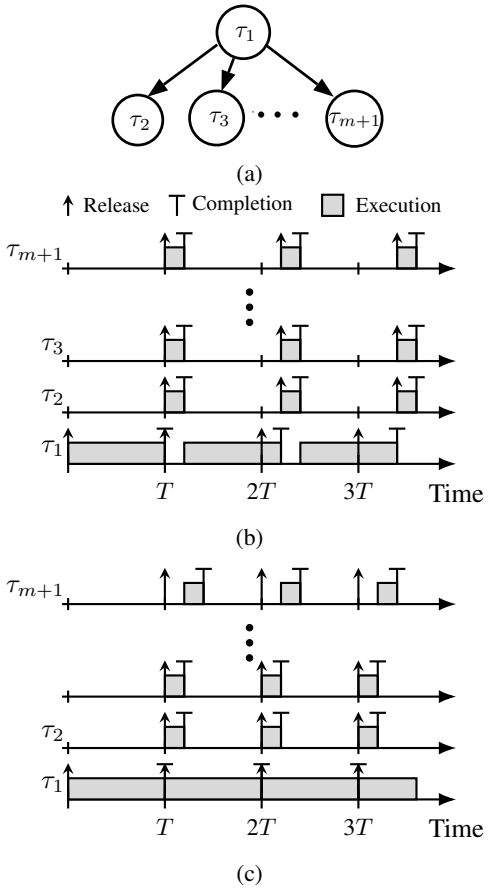


Fig. 3: (a) A DAG G and a schedule of G with (b) unbounded response times and (c) bounded response times.

Brokenness of Lemma 1. The proof of Lemma 1 is not valid when node-level self-dependencies exist. Without node-level self-dependencies, a job $\tau_{i,j}$ in an envelope path cannot execute at a time instant t only because higher-priority jobs occupy all available processors at time t. Unfortunately, this is not true if $P_i < m$, as only P_i prior jobs of $\tau_{i,j}$ (including $\tau_{i,j-P_i}$) may execute at time t, meaning that all available processors may not be busy at time t.

Complexities in bounding carry-in workloads. A common technique to derive response-time bounds is to upper bound the *carry-in* workload that can interfere with a DAG job of interest J_j [12], [28], [42]. Such an upper bound is often determined by considering the latest time instant t_0 before J_j's release when at least one available processor is idle. Without node-level self-dependencies, the number of DAG jobs pending at time t_0 does not exceed the number of busy processors at time t_0. Unfortunately, this does not hold when there are node-

level self-dependencies. For example, in Fig. 3(b), only one processor is busy during $[2T, 2T+1)$, when there are two pending DAG jobs.

V. Scheduling Algorithm

In this section, we give a scheduling algorithm to schedule a DAG G on m processors. We begin by giving an example that motivates our algorithm.

Ex. 4 (Cont'd). Fig. 3(c) depicts a schedule of G in Fig. 3(a). In this schedule, job $\tau_{1,j+1}$ has higher priority than any job $\tau_{i,j}$ with $i \neq 1$. Consequently, each DAG job's response time is at most $T+2$ time units. $\Diamond$

Note that, in Fig. 3(c), a DAG job J_j makes progress towards completion after its release despite the presence of incomplete prior DAG jobs. Our scheduling algorithm is a priority-driven work-conserving one motivated by this insight. We first describe how job priorities are assigned.

Job priorities. We assign each job a fixed *base* priority. However, under our scheduling algorithm, a job's *effective* priority may exceed its base priority. At any time instant, jobs are scheduled according to their effective priorities. We use *priority boosting* to raise a job's effective priority above the highest possible base priority at certain time instants.[4] Thus, a job's effective priority can be either its base priority or an elevated boosted priority.

We let $\tau_{i,j} \prec_b \tau_{k,\ell}$ (resp., $\tau_{i,j} \prec_e \tau_{k,\ell}$) denote that job $\tau_{i,j}$ has a higher base (resp., effective) priority than job $\tau_{k,\ell}$. The following rule specifies conditions for $\tau_{i,j} \prec_b \tau_{k,\ell}$.

BP. At time t, job $\tau_{i,j} \prec_b \tau_{k,\ell}$ holds if and only if $(j < \ell) \vee (j = \ell \wedge i < k)$ holds.

By Rule BP, across different DAG jobs, jobs from earlier DAG jobs have higher base priorities than jobs from later DAG jobs. Within a DAG job, jobs with smaller node indices have higher base priorities than jobs with larger node indices. Recall that, by Assumption I, node indexing follows a topological ordering of G. Thus, each job has a lower priority than all of its ancestors. The following rule specifies conditions for $\tau_{i,j} \prec_e \tau_{k,\ell}$.

EP. At time t, job $\tau_{i,j} \prec_e \tau_{k,\ell}$ holds if and only if *one* of the following three conditions is satisfied.

- **EP1.** At time t, both $\tau_{i,j}$ and $\tau_{k,\ell}$ are priority-boosted and $(j < \ell) \vee (j = \ell \wedge i < k)$ holds.
- **EP2.** At time t, $\tau_{i,j}$ is priority-boosted but $\tau_{k,\ell}$ is not.
- **EP3.** At time t, neither of $\tau_{i,j}$ and $\tau_{k,\ell}$ is priority-boosted and $\tau_{i,j} \prec_b \tau_{k,\ell}$, *i.e.*, $(j < \ell) \vee (j = \ell \wedge i < k)$ holds by Rule BP.

Thus, by Rule EP1, between two jobs with boosted priorities, one receives higher priority if it is from an earlier DAG job, or it has a smaller node index if both jobs are from the same DAG job. By Rule EP2, priority-boosted jobs have higher priorities than non-priority-boosted jobs. By Rule EP3, if two jobs are not priority-boosted, then their effective priority order follows their base priority order.

[4]Priority boosting is also used to ensure a lock-holding job's progress in the context of mutual exclusion locks [34].

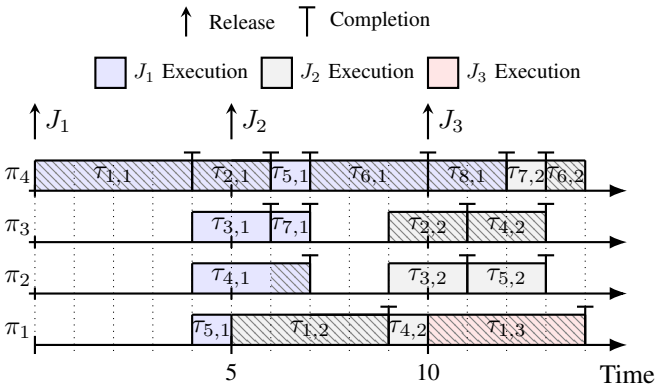


Fig. 4: A schedule of G according to Rule SR when job priorities follow Rules BP, EP, and BO. Hatched rectangles indicate job execution with boosted priorities.

Priority-boosting rules. We now give the rule that specifies when a job is priority-boosted.

BO. At any time instant t, if DAG job J_j is pending, then the *pending* job $\tau_{i,j}$ with the *highest* base priority among all pending jobs of J_j is priority-boosted.

Note that Rule BO does not require a priority-boosted job to be ready. Thus, a priority-boosted job may not be ready to execute due to node-level self-dependencies, as described in Def. 3. From Rule BO, we have the following lemma, which we use in deriving our response-time bounds.

Lemma 2. *If a job $\tau_{i,j}$'s priority is boosted at time t, then it remains so until its completion.*

Proof. By Rule BO, $\tau_{i,j}$ has the highest base priority among all pending jobs of J_j. Thus, by Rule BP, a job $\tau_{k,j}$ is pending at time t if $i < k$ holds, which implies that any job $\tau_{\ell,j}$ with $\ell < i$ completes execution by time t. Thus, $\tau_{i,j}$ continues to have the highest base priority among all pending jobs of J_j until $\tau_{i,j}$ completes. Therefore, by Rule BO, $\tau_{i,j}$'s priority is boosted from time t until its completion. $\square$

Note that Lemma 2 implies that a job priority is elevated at most once during its lifetime.

Scheduling rules. We schedule jobs on m processors in a work-conserving manner according to their effective priorities. Formally, jobs are scheduled according to the following rule.

SR. At any time t, the m jobs (if that many exist) with the highest effective priorities are scheduled.

Ex. 5. Consider the DAG G shown in Fig. 1(a). Assume that $T = 5$. Since $vol(G) = 19$, we have $U = 3.8$. Fig. 4 depicts a schedule of G on four processors $\pi_1, \ldots, \pi_4$. G releases its first three jobs J_1, J_2, and J_3 at times 0, 5, and 10, respectively. At time 0, $\tau_{1,1}$ is the pending job of J_1 with the highest base priority. Therefore, by Rules BO and SR, it is priority-boosted and scheduled. At time 4, $\tau_{1,1}$ completes and $\tau_{2,1}$ becomes the pending job of J_1 with the highest base priority. Thus, $\tau_{2,1}$ is priority-boosted and scheduled at time 4 by Rules BO and SR. At time 5, $\tau_{1,2}$ is released and becomes the pending job of J_2 with the highest base priority and is

priority-boosted. Thus, at time 5, the lowest-priority scheduled job $\tau_{5,1}$ is preempted and $\tau_{1,2}$ is scheduled. At time 6, $\tau_{2,1}$ completes and $\tau_{4,1}$ becomes the pending job of J_1 with the highest base priority. Thus, $\tau_{4,1}$ is priority-boosted at time 6 by Rule BO. $\Diamond$

VI. Response-Time Bound

In this section, we first present a coarse-grained response-time bound (Sec. VI-A), which has a value of $vol(G)$. To establish this bound, we address challenges related to node-level self-dependencies and prove a key property (Lemma 7) concerning the remaining execution time of a job at any time. We later use this property, together with Theorem 2, to derive a fine-grained response-time bound (Sec. VI-B).

A. Coarse-Grained Response-Time Bound

We consider a schedule $\mathcal{S}$ of an arbitrary concrete instantiation of G according to Rule SR when job priorities satisfy Rules BP, EP, and BO. We show that the response time of any DAG job is at most $vol(G)$ (Theorem 3) in $\mathcal{S}$. Intuitively, each DAG job can, in the worst case, execute sequentially while satisfying all node-level self-dependencies, leading to the $vol(G)$ bound. Such execution can be guaranteed due to boosting the priority of one job from each pending DAG job. Therefore, to prove the $vol(G)$ bound, we first show that, at any time, all priority-boosted ready jobs are scheduled (Lemma 6) by showing that there are at most m such jobs (Lemma 5). Using this property, we then upper bound the remaining execution time of each job (Lemma 7), which establishes the $vol(G)$ bound. We prove the bound for an arbitrary DAG job J_q by induction, assuming the following.

A. The response time of each DAG job J_j with $j < q$ is at most $vol(G)$ in $\mathcal{S}$.

We now show that the response time of J_q is also at most $vol(G)$. To show that any priority-boosted ready job is always scheduled, in the following two lemmas, we first upper bound the number of DAG jobs that can be pending at any time.

Lemma 3. *At any time $t \in [0, r(J_q) + mT)$, there are at most $m-1$ pending DAG jobs of G in $\mathcal{S}$ that are released at or before time $t - T$.*

Proof. Let t be a time instant in $[0, r(J_q) + mT)$. Since at most $m - 1$ DAG jobs of G are released before time $(m - 1)T$, for any $t \in [0, mT)$, at most $m - 1$ DAG jobs of G are released at or before time $t - T$. Thus, by Def. 3, the lemma holds if $t < mT$. We now consider $t \geq mT$. We first show that all DAG jobs released at or before time $t - mT$ complete execution by time t. Since $t < r(J_q) + mT$, we have $t - mT < r(J_q)$. Thus, only DAG jobs J_ℓ with $\ell < q$ are released at or before time $t-mT$. By (1), we have $U \leq m$, which implies that $vol(G) \leq mT$. Therefore, by (A), all DAG jobs released at or before time $t-mT$ complete execution by time $t-mT+mT = t$. Therefore, DAG jobs that are pending at time t are released during $(t - mT, t]$. Among all pending DAG jobs at time t, at most $m - 1$ DAG jobs are released during $(t - mT, t - T]$. Thus, by Def. 3, there are at most $m - 1$ pending DAG jobs at time t that are released at or before time $t - T$. □

Lemma 4. *At any time $t \in [0, r(J_q) + mT)$, there are at most m pending DAG jobs of G.*

Proof. By Lemma 3, at time t, there are at most $m-1$ pending DAG jobs of G with release times in $[0, t - T]$. At most one DAG job of G is released during $(t-T, t]$. Thus, there are at most m pending DAG jobs of G at time t. □

Using Lemma 4, the following lemma upper bounds the number of priority-boosted jobs at any time instant.

Lemma 5. *At any time $t \in [0, r(J_q) + mT)$, there are at most m priority-boosted jobs of G in $\mathcal{S}$.*

Proof. At time t, by Rules BO, each pending DAG job of G has one job with boosted priority. By Lemma 4, there are at most m such DAG jobs. Thus, the lemma holds. □

Since each priority-boosted job has higher priority over any non-priority-boosted job, we have the following lemma.

Lemma 6. *At any time $t \in [0, r(J_q) + mT)$, each priority-boosted ready job is scheduled in $\mathcal{S}$.*

Proof. By Lemma 4, there are at most m jobs of G with boosted effective priority at time $t \in [0, r(J_q) + mT)$. By Rules EP2, all such jobs have higher effective priorities than other pending jobs at time t. By Rule SR, each of the jobs with boosted effective priority is scheduled at time t if it is ready. Thus, the lemma holds. □

We now upper bound the remaining execution time of a job at any time instant. Upper bounding this term allows us to directly prove our bound of $vol(G)$. We first give a notation to denote the remaining execution time of a job.

Def. 6. *Let $C_{i,j}(t)$ be the remaining execution time of job $\tau_{i,j}$ at time t in $\mathcal{S}$.*

We upper bound $C_{i,j}(t)$ using the notation introduced next.

Def. 7. *Let V_k denote the set of tasks $\{\tau_1, \tau_2, \ldots, \tau_k\}$ of DAG G and $vol(V_k) = \sum_{i=1}^{k} C_i$. We let $V_0 = \emptyset$ and $vol(V_0) = 0$.*

By Def. 7, the following holds.

$$\forall\ 1 \leq k \leq n : vol(V_k) = vol(V_{k-1}) + C_k \tag{4}$$

Note that $V_n = V$ holds. Thus, by Def. 2, $vol(G) = vol(V_n)$. In the following lemma, we upper bound $C_{i,j}(t)$ values by considering job execution by Rule SR in the presence of node-level self-dependencies.

Lemma 7. *For any job $\tau_{i,j}$ with $j \leq q$ and $t \in [0, r(J_q)+mT)$,*

$$C_{i,j}(t) \leq \min\{\max\{vol(V_i) - (t - r(J_j)), 0\}, C_i\}. \tag{5}$$

Proof. Fig. 5 illustrates the proof. Assume otherwise. Then, let $t \in [0, r(J_q)+mT)$ be the first time instant such that there exists a job for which (5) does not hold at time t. Let $\tau_{i,j}$ be the job with the highest base priority for which (5) does not hold at time t. Thus, we have

$$C_{i,j}(t) > \min\{\max\{vol(V_i) - (t - r(J_j)), 0\}, C_i\}, \tag{6}$$

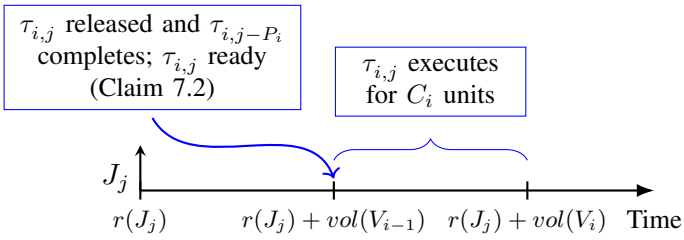


Fig. 5: Illustration of Lemma 7

and

$$(\forall t' < t, \tau_{k,\ell} : C_{k,\ell}(t') \leq \min\{\max\{vol(V_k) - (t' - r(J_\ell)), 0\}, C_k\}) \bigwedge (\forall \tau_{k,\ell} \prec_b \tau_{i,j} : C_{k,\ell}(t) \leq \min\{\max\{vol(V_k) - (t - r(J_\ell)), 0\}, C_k\}) \quad (7)$$

By (6), we have $C_{i,j}(t) > 0$. Therefore, we have

$$f(\tau_{i,j}) > t. \quad (8)$$

We now show that the following holds.

$$t \geq r(J_j) + vol(V_{i-1}) \quad (9)$$

To see that (9) holds note that, if $t < r(J_j) + vol(V_{i-1})$ holds, then we have $vol(V_i) - (t - r(J_j)) > vol(V_i) - (r(J_j) + vol(V_{i-1}) - r(J_j)) = vol(V_i) - vol(V_{i-1}) = C_i$ (by Def. 7 and (4)). This implies that $\max\{vol(V_i) - (t - r(J_j)), 0\} > C_i$. Therefore, by (6), $C_{i,j}(t) > \min\{\max\{vol(V_i) - (t - r(J_j)), 0\}, C_i\} > \min\{C_i, C_i\} = C_i$. However, by Def. 6, $C_{i,j}(t) \leq C_i$ holds. Thus, $t < r(J_j) + vol(V_{i-1})$ cannot hold.

We now prove the lemma by first showing that $\tau_{i,j}$ is released by time $r(J_j) + vol(V_{i-1})$. We then show that $\tau_{i,j}$ is ready to execute (by Def. 3) by time $r(J_j) + vol(V_{i-1})$. Finally, we show that $\tau_{i,j}$ has high-enough priority to be scheduled at or after time $r(J_j) + vol(V_{i-1})$ until it completes execution. This would imply that $\tau_{i,j}$ executes for enough time so that (6) does not hold, reaching a contradiction. To prove $\tau_{i,j}$ is released by time $r(J_j) + vol(V_{i-1})$, in the following claim, we show that all predecessor jobs of $\tau_{i,j}$ completes execution by time $r(J_j) + vol(V_{i-1})$.

Claim 7.1. *Each job $\tau_{k,j}$ with $k < i$ finishes execution by time $r(J_j) + vol(V_{i-1})$ in $\mathcal{S}$.*

Proof. By (4), we have $r(J_j) + vol(V_{i-1}) = r(J_j) + vol(V_i) - C_i \leq r(J_j) + vol(V_i)$, as $C_i \geq 0$. By Rule BP, $\tau_{k,j} \prec_b \tau_{i,j}$. Thus, by (7) and since $t \geq r(J_j) + vol(V_{i-1})$ (by (9)), we have $C_{k,j}(r(J_j) + vol(V_{i-1})) \leq \min\{\max\{vol(V_k) - (r(J_j) + vol(V_{i-1}) - r(J_j)), 0\}, C_k\} = \min\{\max\{vol(V_k) - vol(V_{i-1}), 0\}, C_k\}$. Since $k < i$, by (4), we have $vol(V_k) \leq vol(V_{i-1})$. Therefore, $C_{k,j}(r(J_j) + vol(V_{i-1})) \leq \min\{0, C_k\} = 0$. By Def. 6, no execution of $\tau_{k,j}$ is remaining at time $r(J_j) + vol(V_{i-1})$. Therefore, $\tau_{k,j}$ finishes execution by time $r(J_j) + vol(V_{i-1})$. □

Recall that task indices follow a topological ordering of G. By Claim 7.1, all jobs $\tau_{k,j}$ with $k < i$ complete execution by time $r(J_j) + vol(V_{i-1})$. Therefore, we have

$$r(\tau_{i,j}) \leq r(J_j) + vol(V_{i-1}). \quad (10)$$

We now show that $\tau_{i,j}$ is ready to execute by time $r(J_j) + vol(V_{i-1})$. This requires showing that the job of τ_i on which $\tau_{i,j}$ depends on completes execution by time $r(J_j) + vol(V_{i-1})$.

Claim 7.2. *Job $\tau_{i,j}$ is ready at time $r(J_j) + vol(V_{i-1})$ in $\mathcal{S}$.*

Proof. By (10), job $\tau_{i,j}$ is released by time $r(J_j) + vol(V_{i-1})$. By (8) and (9), $f(\tau_{i,j}) > t \geq r(J_j) + vol(V_{i-1})$. Therefore, by Def. 3, job $\tau_{i,j}$ is ready at time $r(J_j) + vol(V_{i-1})$ if $j \leq P_i$ or job $\tau_{i,j-P_i}$ (if one exists) completes execution by time $r(J_j) + vol(V_{i-1})$. The case with $j \leq P_i$ is trivial, so we assume $j > P_i$. Let $t' = r(J_{j-P_i}) + vol(V_i)$. Since DAG jobs are released sporadically, we have $t' \leq r(J_j) - P_iT + vol(V_i) = r(J_j) - P_iT + vol(V_{i-1}) + C_i$. By (1), we have $u_i = C_i/T \leq P_i$. Thus, we have $t' \leq r(J_j) - P_iT + vol(V_{i-1}) + P_iT = r(J_j) + vol(V_{i-1})$. Thus, by (9), we have

$$t' \leq r(J_j) + vol(V_{i-1}) \leq t. \quad (11)$$

By Rule BP, $\tau_{i,j-P_i} \prec_b \tau_{i,j}$. Thus, by (7), $C_{i,j-P_i}(t') \leq \min\{\max\{vol(V_i) - (t' - r(J_{j-P_i})), 0\}, C_i\} = \min\{\max\{vol(V_i) - (r(J_{j-P_i}) + vol(V_i) - r(J_{j-P_i})), 0\}, C_i\} = \min\{\max\{vol(V_i) - vol(V_i), 0\}, C_i\} = 0$. Thus, $\tau_{i,j-P_i}$ completes execution by time t'. Therefore, by (11), $\tau_{i,j}$ is ready at time $r(J_j) + vol(V_{i-1})$. □

We now complete the proof by showing that $\tau_{i,j}$ is scheduled during $[r(J_j) + vol(V_{i-1}), t)$. By Claim 7.2, $\tau_{i,j}$ is ready (hence, pending) at time $r(J_j) + vol(V_{i-1})$. By Claim 7.1, among all pending jobs of J_j at time $r(J_j) + vol(V_{i-1})$, job $\tau_{i,j}$ has the smallest node index, hence the highest base priority (by Rule BP). Therefore, by Rule BO and Lemma 2, $\tau_{i,j}$'s priority is boosted at $r(J_j) + vol(V_{i-1})$ and remains so until its completion. Thus, by Lemma 6, $\tau_{i,j}$ is scheduled at $r(J_j) + vol(V_{i-1})$ and remains so during $[r(J_j) + vol(V_{i-1}), t)$. Therefore, $\tau_{i,j}$ executes for at least $t - r(J_j) - vol(V_{i-1})$ time units by time t. Thus, $C_{i,j}(t) \leq C_i - (t - r(J_j) - vol(V_{i-1})) = vol(V_i) - (t - r(J_j)) \leq \max\{vol(V_i) - (t - r(J_j)), 0\}$. By (9), $t - r(J_j) \geq vol(V_{i-1})$, which implies that $vol(V_i) - (t - r(J_j)) \leq vol(V_i) - vol(V_{i-1}) = C_i$. Therefore, $\max\{vol(V_i) - (t - r(J_j)), 0\} \leq C_i$. Thus, $C_{i,j}(t) \leq \max\{vol(V_i) - (t - r(J_j)), 0\} \leq \min\{\max\{vol(V_i) - (t - r(J_j)), 0\}, C_i\}$, contradicting (6). □

Using Lemma 7, we now prove the following lemma, which by induction proves Theorem 3.

Lemma 8. *The response time $R(J_q)$ of DAG job J_q is at most $vol(G)$ in $\mathcal{S}$.*

Proof. Let τ_i be the node of G with the largest node index such that $C_i > 0$. Recall that C_n can be zero if the sink is virtual. By (1), $U = vol(V_n)/T \leq m$ holds. Therefore, $vol(V_n) \leq mT$. Since $C_k = 0$ holds for any $k \in \{i + 1, \ldots, n\}$, we have $vol(V_i) = vol(V_n) \leq mT$. For any ϵ satisfying $0 < \epsilon < C_i$, we have $r(J_q) + vol(V_i) - \epsilon < r(J_q) + mT$. By Lemma 7, $C_{i,q}(r(J_q) + vol(V_i) - \epsilon) = \epsilon$

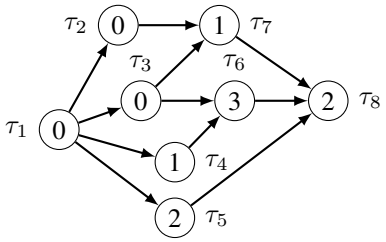


Fig. 6: $G(2)$ corresponding to G in Fig. 1 (numbers inside circles represent WCETs).

and $C_{\ell,q}(r(J_q) + vol(V_i) - \epsilon) = 0$ for all $\ell < i$. Thus, $\tau_{i,q}$ is the pending job of J_q with the highest base priority and is ready (as it has already executed for $C_i - \epsilon$ time units). Thus, it is scheduled at time $r(J_q) + vol(V_i) - \epsilon$ until it completes by time $r(J_q) + vol(V_i) = r(J_q) + vol(V_n)$ (by Lemma 6). Since each job $\tau_{\ell,q}$ with $\ell > i$ has $C_\ell = 0$, J_q completes by time $r(J_q) + vol(V_n)$. Therefore, J_q's response time is at most $vol(V_n) = vol(G)$. □

Theorem 3. *The response time of G in $\mathcal{S}$ is at most $vol(G)$.*

Proof. Follows from Lemma 8 and Assumption A. □

B. Fine-Grained Response-Time Bound

In this section, we improve the bound presented in Theorem 3 using Theorem 2. To derive our bound, we consider the schedule $\mathcal{S}$, as defined in Sec. VI-A. We begin by defining a DAG $G(\ell)$. Intuitively, in $G(\ell)$, each node's WCET is an upper bound on the remaining execution time of any DAG job J_j at time $r(J_j) + \ell T$ in $\mathcal{S}$ (Lemma 12 gives a formal proof).

Def. 8. *For any integer $0 \leq \ell \leq m$, let $G(\ell)$ be a DAG with the same nodes V and edges E in G. The WCET $C_i(\ell)$ of each node τ_i of $G(\ell)$ is defined as follows.*

$$C_i(\ell) = \begin{cases} 0 & \text{if } vol(V_i) \leq \ell T \\ vol(V_i) - \ell T & \text{if } vol(V_{i-1}) \leq \ell T < vol(V_i) \\ C_i & \text{otherwise} \end{cases} \tag{12}$$

Ex. 5 (Cont'd). Consider the DAG G in Fig. 1(a). Fig. 6 illustrates $G(2)$ for $T = 5$. We have $2T = 10$. Since $vol(V_1) = 4$, $vol(V_2) = 6$, and $vol(V_3) = 8$, by the first case of (12), $C_1(2) = C_2(2) = C_3(2) = 0$. Since $vol(V_4) = 11$ and $vol(V_3) = 8$, by the second case of (12), $C_4(2) = 11 - 10 = 1$. Since $vol(V_5) = 13$ and $vol(V_4) = 11$, by the third case of (12), $C_5(2) = 2$. ◊

The following lemma shows that $C_i(\ell)$ upper bounds the remaining execution time of any job $\tau_{i,j}$ at time $r(J_j) + \ell T$.

Lemma 9. *For any i, j, and $\ell \geq 1$, $C_{i,j}(r(J_j) + \ell T) \leq C_i(\ell)$ holds in $\mathcal{S}$, where $C_{i,j}(r(J_j) + \ell T)$ is defined by Def. 6.*

Proof. By Lemma 7, we have

$$\begin{aligned} &C_{i,j}(r(J_j) + \ell T) \\ &\leq \min\{\max\{vol(V_i) - (r(J_j) + \ell T - r(J_j)), 0\}, C_i\} \\ &= \min\{\max\{vol(V_i) - \ell T, 0\}, C_i\}. \end{aligned} \tag{13}$$

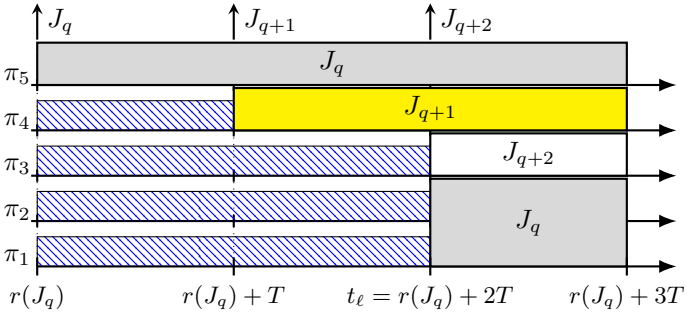


Fig. 7: Proof of Lemma 10 for $m = 5$ and $\ell = 2$. Execution of J_q (resp., J_{q+1} and J_{q+2}) is shaded gray (resp., yellow and white). Jobs of DAG jobs prior to J_q execute during the hatched rectangles.

Note that Lemma 7 depends on Assumption A, which holds due to Theorem 3. We now consider three cases based on (12).

Case 1. $vol(V_i) \leq \ell T$. Thus, $\max\{vol(V_i) - \ell T, 0\} = 0$. Hence, by (13), $C_{i,j}(r(J_j) + \ell T) \leq \min\{0, C_i\} = 0$ holds.

Case 2. $vol(V_{i-1}) \leq \ell T < vol(V_i)$. Therefore, $vol(V_i) - \ell T > 0$, which implies $\max\{vol(V_i) - \ell T, 0\} = vol(V_i) - \ell T$. Therefore, $\min\{\max\{vol(V_i) - \ell T, 0\}, C_i\} = \min\{vol(V_i) - \ell T, C_i\}$, which by (13) implies $C_{i,j}(r(J_j) + \ell T) \leq \min\{vol(V_i) - \ell T, C_i\}$.

Since $\ell T \geq vol(V_{i-1})$, we have $vol(V_i) - \ell T \leq vol(V_i) - vol(V_{i-1}) = C_i$ (by (4)). Thus, we have $C_{i,j}(r(J_j) + \ell T) \leq \min\{vol(V_i) - \ell T, C_i\} = vol(V_i) - \ell T$.

Case 3. By Def. 6, $C_{i,j}(r(J_j) + \ell T) \leq C_i$ holds.

Thus, the lemma holds for all three cases. □

We now derive a response-time bound for G in $\mathcal{S}$ by proving the following lemma. We use Fig. 7 to illustrate the proof.

Lemma 10. *Let $\ell \in \{0, 1, \ldots, m-1\}$ and $p \in \{1, 2, \ldots, m - \ell\}$ be two integers, and $(\lambda_i(\ell))_1^p$ be a generalized path list of $G(\ell)$ such that $\lambda_1(\ell)$ is the longest path in $G(\ell)$. Then, the response time of $G(\ell)$ is bounded by $\ell T + R^\ell$ if $R^\ell \leq T$ holds, where*

$$R^\ell = \min_{1 \leq j \leq p} \{len(G(\ell)) + \frac{vol(G(\ell)) - \sum_{i=1}^{j} len(\lambda_i(\ell))}{m - \ell - j + 1}\}. \tag{14}$$

To prove Lemma 10, we assume that there exist ℓ and p, as defined in the statement of Lemma 10, such that $R^\ell \leq T$ holds. We consider a DAG job of interest J_q and inductively show that J_q's response time is at most $\ell T + R^\ell$. To do so, we assume the following.

B. The response time of each DAG job J_j of G with $j < q$ is bounded by $\ell T + R^\ell$.

Let $t_\ell = r(J_q) + \ell T$. Informally, we show that all DAG jobs prior to J_q complete execution by time t_ℓ (Lemma 11). Consequently, all self-dependencies for jobs of J_q are satisfied by time t_ℓ. Furthermore, only DAG jobs released after J_q can interfere with J_q at or after t_ℓ, and the number of such DAG jobs is at most ℓ (Lemma 13). Since each DAG job other than J_q can have only one priority-boosted ready job, they occupy

at most ℓ processors, leaving $m-\ell$ processors available for J_q (see Fig. 7). Note that R^ℓ is a response-time bound of $G(\ell)$ on $m-\ell$ processors according to Theorem 2. Also, since at most m DAG jobs are pending at any time (Lemma 4), $\ell \leq m-1$.

Lemma 11. *Each DAG job J_j with $j < q$ finishes by time t_ℓ.*

Proof. By Assumption B, J_j finishes by time $r(J_j)+\ell T+R^\ell$. Since $R^\ell \leq T$, we have $r(J_j)+\ell T+R^\ell \leq r(J_j)+(\ell+1)T$. Since $j < q$, we have $r(J_j) \leq r(J_q)-T$. Thus, $r(J_j)+(\ell+1)T \leq r(J_q)+\ell T = t_\ell$. Therefore, the lemma holds. □

Since all DAG jobs prior to J_q finish execution by time t_ℓ, no jobs of J_q are delayed due to node-level self-dependencies at or after t_ℓ. Thus, jobs of J_q are ready to execute at or after t_ℓ immediately upon release, as shown below.

Lemma 12. *At any time $t \in [t_\ell, t_\ell+T)$, if $r(\tau_{i,q}) \leq t < f(r(\tau_{i,q}))$, then $\tau_{i,q}$ is ready at time t.*

Proof. By Def. 3, $\tau_{i,q}$ is ready at time $t \in [r(\tau_{i,q}), f(r(\tau_{i,q}))$ if $f(\tau_{i,j-P_i}) \leq t$ (if $j > P_i$) holds. By Lemma 11, $f(\tau_{i,j-P_i}) \leq t_\ell \leq t$ (if $j > P_i$) holds. Therefore, $\tau_{i,q}$ is ready at time t. □

The following lemma upper bounds the number of priority-boosted jobs of DAG jobs J_j with $j > q$ at any time $t \in [t_\ell, t_\ell+T)$. This allows us to lower-bound the number of processors available for the execution of J_q (Lemma 15) when a sufficiently large number of its jobs are ready.

Lemma 13. *At any time instant $t \in [t_\ell, t_\ell+T)$, at most ℓ jobs of DAG jobs other than J_q are priority-boosted.*

Proof. By Lemma 11, no DAG jobs with $j < q$ are pending at or after time t_ℓ. Thus, only DAG jobs with $j \geq q$ that are released at or before $t \geq t_\ell$ can be pending at time t. Therefore, DAG jobs that are pending at time t are released during the interval $[r(J_q), t)$. Since J_q is released at $r(J_q)$, $t_\ell = r(J_q)+\ell T$, and $t \in [t_\ell, t_\ell+T)$, the number of DAG jobs other than J_q released during such an interval is at most ℓ. By Rule BO, for each pending DAG job at time t, only the pending job with the highest base priority is priority-boosted. Therefore, there are at most ℓ jobs of DAG jobs other than J_q that are priority-boosted. □

The following lemma shows that, at or after t_ℓ, J_q's ready jobs have higher effective priorities than any non-priority-boosted ready jobs of other pending DAG jobs.

Lemma 14. *Let $\tau_{i,q}$ and $\tau_{k,j}$ be two ready jobs with $q \neq j$ at time $t \in [t_\ell, t_\ell+T)$. If $\tau_{k,j}$ is not priority-boosted at time t, then $\tau_{i,q} \prec_e \tau_{k,j}$.*

Proof. Since $q \neq j$, by Lemma 11, $q < j$ holds. If $\tau_{i,q}$ is priority-boosted at time t, then by Rule EP2, $\tau_{i,q} \prec_e \tau_{k,j}$. In contrast, if $\tau_{i,q}$ is not priority-boosted at time t, then by Rule EP3, $\tau_{i,q} \prec_e \tau_{k,j}$. Thus, the lemma holds. □

By Lemmas 13 and 14, at least $m-\ell$ ready jobs of J_q (if that many exist) execute at any time $t \in [t_\ell, t_\ell+T)$. The following lemma shows this.

Lemma 15. *Assume that there are g ready jobs of J_q at time $t \in [t_\ell, t_\ell+T)$. Then, at least $\min\{g, m-\ell\}$ jobs of J_q with the highest base priorities among the g ready jobs of J_q are scheduled at time t.*

Proof. By Lemma 13, at time t, there are at most ℓ jobs of DAG jobs other than J_q with boosted priority. By Rule BO, there is at most one ready job of J_q with boosted priority at time t. Since we assume $0 \leq \ell \leq m-1$ (Lemma 10 statement), by Rule EP2, all ready jobs with boosted priorities have higher effective priorities than other ready jobs at time t, hence are scheduled. Thus, at time t, there are at most ℓ processors that execute priority-boosted jobs of DAG jobs other than J_q.

Let Π be the set of processors that do not execute priority-boosted jobs of DAG jobs other than J_q at time t. We have $|\Pi| \geq m-\ell$. By Rule BO, the priority-boosted job of J_q at time t is the pending job of J_q with the highest base priority. By Lemma 14, all ready jobs of J_q have higher effective priorities than any non-priority-boosted job of J_j with $j > q$. Thus, by Rule SR, $\min\{g, |\Pi|\}$, hence at least $\min\{g, m-\ell\}$ ready highest-base-priority jobs of J_q are scheduled at time t. □

Thus, the execution of J_q's jobs during $[t_\ell, t_\ell+T)$ is equivalent to its execution on at least $m-\ell$ processors in isolation, *i.e.*, ignoring all jobs of other DAG jobs (see Fig. 7). The following definition gives a schedule of a DAG job on $m-\ell$ processors whose workload upper bounds J_q's workload during $[t_\ell, t_\ell+T)$.

Def. 9. *Let $J(\ell)$ be a non-recurrent DAG job of $G(\ell)$. Let $\tau_i(\ell)$ be the job of τ_i in $J(\ell)$, and $C(\tau_i(\ell)) = C_{i,q}(t_\ell)$ be the execution time of $\tau_i(\ell)$. Note that, by Lemma 9, $C(\tau_i(\ell)) \leq C_i(\ell)$, thus, execution times of $J(\ell)$'s jobs are valid. Let $\mathcal{S}_\ell$ be a schedule of $J(\ell)$ on $m-\ell$ processors according to Rule SR when job priorities are determined by Rules BP, SP, and BO.*

Lemma 16. *The response time of $J(\ell)$ is at most R^ℓ in $\mathcal{S}_\ell$.*

Proof. Since $\mathcal{S}_\ell$ is work-conserving due to Rule SR, the lemma follows from Theorem 2. □

We now prove Lemma 10.

Proof of Lemma 10. By Lemma 12, each job $\tau_{i,q}$ of J_q that is pending at time t_ℓ becomes ready by time $\min\{r(\tau_{i,q}), t_\ell\}$. In $\mathcal{S}_\ell$, each job in $J(\ell)$ is also ready as soon as it is released, as $J(\ell)$ is a non-recurrent instantiation of $G(\ell)$. By Def. 9, the remaining execution time of each job in $\mathcal{S}$ at time t_ℓ is equal to the execution time of $\tau_i(\ell)$ in $\mathcal{S}_\ell$. By Lemma 15, J_q executes on at least $m-\ell$ processors during $[t_\ell, t_\ell+T)$. Thus, since $J(\ell)$'s response time is at most R^ℓ (Lemma 16) and $R^\ell \leq T$ (by the assumption of Lemma 10), J_q completes execution by time $t_\ell+R^\ell$. (Note that, since replacing $m-\ell$ by a larger quantity decreases the r.h.s. of (14), R^ℓ is still a valid bound if the number of processors is greater than $m-\ell$.) Thus, J_q's response time is at most $\ell T+R^\ell$. □

Note that Lemma 10 requires $R^\ell \leq T$. In the following lemma, we show the existence of an ℓ that satisfies $R^\ell \leq T$.

Lemma 17. *$R^{m-1} \leq T$ holds, where R^{m-1} is defined by* (14).

Proof. By (12), for each task τ_i with $vol(V_i) \leq (m-1)T$, $C_i(m-1) = 0$ holds. Let τ_k be the task with the smallest k such that $vol(V_k) > (m-1)T$. Thus, $vol(V_{k-1}) \leq (m-1)T$ holds. By (12), we have $C_k(m-1) = vol(V_k) - (m-1)T$. By (12), for any $i > k$, we have $C_i(m-1) = C_i$. Therefore, by Def. 2, we have

$$\begin{aligned}
& vol(G(m-1)) \\
&= \sum_{i=1}^{k-1} C_i(m-1) + C_k(m-1) + \sum_{i=k+1}^{n} C_i(m-1) \\
&= vol(V_k) - (m-1)T + \sum_{i=k+1}^{n} C_i \\
&= vol(V_k) - (m-1)T + vol(V_n) - vol(V_k) \\
&= vol(V_n) - (m-1)T \\
&\leq \{\text{Since } U \leq m \text{ by (1)}\} \\
&\quad mT - (m-1)T \\
&= T. \qquad (15)
\end{aligned}$$

We now determine R^{m-1} by (14). Since $\ell = m-1$, we have $m - \ell = 1$. Therefore, by the assumption of Lemma 10, $1 \leq p \leq m - \ell = 1$. Thus, the only value of j to consider in (14) is 1. For $j = 1$, we have

$$\begin{aligned}
& R^{m-1} \\
=& len(G(m-1)) + \frac{vol(G(m-1)) - \sum_{i=1}^{1} len(\lambda_i(m-1))}{m - (m-1) - 1 + 1} \\
=& \{\text{Since } \lambda_1(m-1) \text{ is the longest path of } G(m-1)\} \\
& len(G(m-1)) + vol(G(m-1)) - len(G(m-1)) \\
\leq& \{\text{By (15)}\} \\
& T.
\end{aligned}$$

Thus, the lemma holds. □

We now have the following theorem.

Theorem 4. *Let $\ell \in \{0, 1, \ldots, m-1\}$ be the smallest integer such that $R^\ell \leq T$ holds, where R^ℓ is defined as in* (14)*. Then, G's response time is at most $\ell T + R^\ell$.*

Proof. By Lemma 17, ℓ exists. By Lemma 10, the theorem holds. □

Response-time bound computation. The algorithm for computing the response-time bound in Theorem 4 is presented in Algorithm 1. Lines 2 and 3 check SRT-feasibility of DAG G and return if it is not. The for-loop in lines 4–11 searches for the smallest non-negative integer ℓ such that $R^\ell \leq T$ holds, and returns $\ell T + R^\ell$ once such an ℓ is found. Lines 5 and 6 compute $G(\ell)$ and a generalized path list of $G(\ell)$ according to Def. 8 and Algorithm 2 in [16], respectively. Lines 7 and 8 then determine a subset of generalized paths containing at most $m - \ell$ paths, including the longest path in $G(\ell)$. Finally, lines 9–11 compute R^ℓ using (14) and check whether the bound in Theorem 4 has been found.

Algorithm 1 Fine-grained response-time bound computation.

Variables:
G : A DAG
m : Number of processors

```
1:  procedure FINE-GRAINED RESPONSE-TIME BOUND
2:      if (1) is not satisfied then
3:          return unbounded response time
4:      for each ℓ = 0 to m − 1 do
5:          G(ℓ) ← DAG by Def. 8
6:          (λ_i(ℓ))_1^{p'} ← Generalized path list of G(ℓ) by [16, Alg. 2]
7:          p ← min{m − ℓ, p'}
8:          (λ_i(ℓ))_1^p ← p paths of (λ_i(ℓ))_1^{p'} with λ_1(ℓ) = len(G(ℓ))
9:          Compute R^ℓ by (14)
10:         if R^ℓ ≤ T then
11:             return ℓT + R^ℓ
```

Time complexity. For any DAG G, a generalized path list satisfying the constraints specified in Lemma 10 can be determined in $O(V(V+E))$ time by methods from [16]. Thus, since $p \leq m$ in (14), after computing a generalized path list of $G(\ell)$, an additional $O(m)$ time is needed to compute R^ℓ. Thus, each R^ℓ can be computed in $O(m) + O(V(V+E))$ time. By Theorem 4, the number of required R^ℓ values is at most m. Thus, the overall time complexity to compute the bound in Theorem 4 is $O(m^2) + O(mV(V+E))$.

Dominance of the fine-grained bound. The following theorem shows that the fine-grained response-time bound in Theorem 4 is not larger than the one in Theorem 3.

Theorem 5. *Let R_f and R_c be the response-time bounds of G by Theorems 3 and 4, respectively, when G is scheduled on m processors by Rule SR and job priorities are determined by Rules BP, EP, and BO. Then, $R_f \leq R_c$ holds.*

Proof. By Theorem 4, $R_f = \ell T + R^\ell$ where R^ℓ is defined as in (14) and ℓ is the smallest integer between 0 and $m-1$ such that $R^\ell \leq T$ holds. By (14), we have

$$\begin{aligned}
R^\ell &= \min_{1 \leq j \leq p} \{len(G(\ell)) + \frac{vol(G(\ell)) - \sum_{i=1}^{j} len(\lambda_i(\ell))}{m - \ell - j + 1}\} \\
&\leq \{\text{Using } j = 1\} \\
&\quad len(G(\ell)) + \frac{vol(G(\ell)) - len(\lambda_1(\ell))}{m - \ell - 1 + 1} \\
&\leq \{\text{Since } \ell \leq m-1 \text{ and } \lambda_1(\ell) \text{ is the longest path of } G(\ell)\} \\
&\quad len(G(\ell)) + vol(G(\ell)) - len(G(\ell)) \\
&= vol(G(\ell)). \qquad (16)
\end{aligned}$$

Let k be the largest integer such that $vol(V_k) \leq \ell T$. Therefore, by (12), $C_i(\ell) = 0$ for all $i \leq k$, $C_{k+1}(\ell) = vol(V_{k+1}) - \ell T$, and $C_i(\ell) = C_i$ for all $i > k+1$. Therefore, we have $vol(G(\ell)) = vol(V_{k+1}) - \ell T + \sum_{i=k+2}^{n} C_i = vol(V_{k+1}) - \ell T + vol(V_n) - vol(V_{k+1}) = vol(G) - \ell T$. By (16), we have $R_f = \ell T + R^\ell \leq \ell T + vol(G(\ell)) \leq \ell T + vol(G) - \ell T = vol(G) = R_c$ (by Theorem 3). Thus, the theorem holds. □

Discussion. The response-time bounds presented in Theorems 3 and 4 do not depend on the P_i values. Thus, the same bound applies for any arbitrary assignment of P_i values. We

achieve this by only analyzing a later portion of the schedule before which node-level self-dependencies are met, *i.e.*, such dependencies are met during $[t_\ell, t_\ell + T)$ by Lemma 11. Our bound is *sustainable*, meaning that the response-time bound is not violated if some job executes for less than its WCET. (In deriving the response-time bound, we did not assume any job to execute for its full WCET.) Moreover, the response-time bounds in Theorems 3 and 4 can be used to perform a schedulability test if the DAG's deadline is hard.

If Theorem 4 is satisfied for $\ell = 0$, then each DAG job finishes before the next DAG job is released. In such a case, our scheduler works as a *job-level-fixed-priority* scheduler, as elevating the highest-base-priority pending job among all pending jobs has no effect. Furthermore, the response-time bound in Theorem 4 matches the one in Theorem 2.

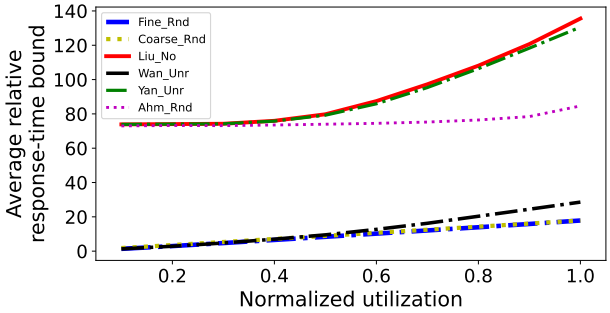
Finally, our scheduling algorithm may incur additional overhead, thus causing more *overhead-induced capacity loss*, compared to job-level-fixed-priority schedulers due to priority boosting rules. The overhead associated with priority boosting can be accounted for by inflating node WCETs. However, recall that each job's priority is boosted at most once.

Multiple DAG tasks. Our scheduling algorithm and response-time analysis can be applied to multi-DAG settings under *federated scheduling*, where each DAG is allocated a set of distinct processors. However, doing so does not preserve soft-real-time optimality. This is due to partitioning-related utilization loss. Thus, to maintain SRT-optimality, each DAG should be scheduled on a set of fully available processors along with some shared processors that are only partially available to the DAG. Dealing with such fractional capacities becomes challenging in the presence of node-level self-dependencies. In contrast, decomposition-based global scheduling approaches can provide SRT-optimality [4]. However, devising a non-decomposition-based approach ensuring no utilization loss becomes complicated due to node-level self-dependencies. Furthermore, since different DAGs can have different periods, techniques presented in this paper do not directly apply.

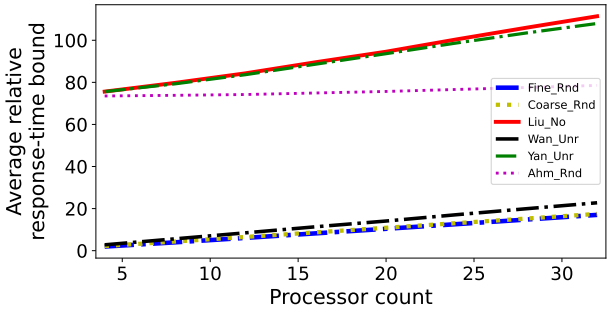
VII. Evaluation

We now present the results of experiments we conducted to evaluate the response-time bounds of our proposed scheduler. We compared it with other soft-real-time optimal schedulers that provide bounded response times with no utilization loss.

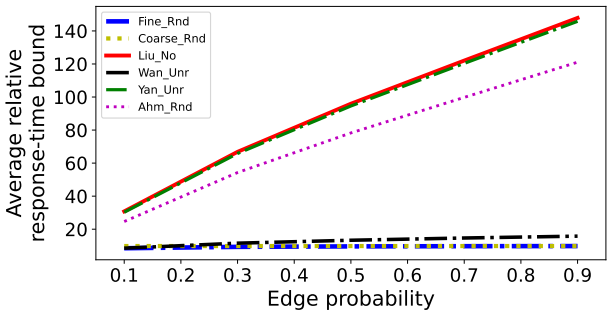
We randomly generated task systems for systems with 4 to 32 processors with a step size of 4.0. Such processor counts are common in real-world use cases [3], [21]. For each processor count, we generated task systems consisting of a single DAG with *normalized utilization*, *i.e.*, U/m, from 0.1 to 1 with a step size of 0.1. Each DAG's period was uniformly selected from $\{1, 2, 5, 10, 20, 50, 100, 200\}$ms [22]. The number of nodes was chosen uniformly from $[10, 100)$. Each node's utilization was chosen uniformly following procedures from [11]. The WCET of each node was rounded to the nearest microsecond. The P_i value of each node was chosen uniformly from $[1, m]$. Edges were generated following the *Erdős-Rényi method* [10]. For each pair of nodes (τ_i, τ_j) with $i < j$, an edge from τ_i to τ_j was added if a uniformly generated number in $[0, 1]$ was at most a predefined *edge-generation probability*. We selected this probability value from $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. As in [35], a minimum number of additional edges was added to make each DAG weakly connected. For each combination of processor count, normalized utilization, and edge-generation probability, we generated 1,000 random task systems.



(a) Resp.-time bound vs. normalized util.



(b) Resp.-time bound vs. proc. count.



(c) Resp.-time bound vs. edge-gen. prob.

Fig. 8: Experimental results.

We compared our *relative response-time bounds* (*i.e.*, response-time bounds normalized by DAG periods) FINE-RND (Theorem 4) and COARSE-RND (Theorem 3) with LIU-NO [25], WAN-UNR [42], YAN-UNR [43], and AHM-RND [1].[5] LIU-NO and YAN-UNR schedule DAGs by assigning offsets and deadlines to nodes and then using GEDF to schedule nodes by their assigned deadlines. AHM-RND schedules DAGs by per-node GEDF-scheduled reservation servers

[5]We did not compare against the bounds in [4], which support arbitrary P_i values, because they are minimized for $P_i = \infty$, equaling YAN-UNR.

and requires a pseudo-polynomial-time (exponential-time for multiple DAGs) algorithm to determine exact response-time bounds. WAN-UNR schedules DAGs under GEDF according to the deadlines of DAG jobs. Among these four approaches, only WAN-UNR provides a non-decomposition-based analysis of DAGs. The suffixes UNR, NO, and RND mean that the corresponding bound assumes unrestricted ($P_i = \infty$), no parallelism ($P_i = 1$), and randomly generated P_i values, respectively.[6] Recall that our bound applies for arbitrary P_i values, but the bound itself does not depend on P_i. The response-time bounds are plotted in Fig. 8(a)–(c).

Observation 1. *On average,* FINE-RND *was 0.9, 0.16, 0.85, 0.16, and 0.23 times* COARSE-RND*,* LIU-NO*,* WAN-UNR*,* YAN-UNR*, and* AHM-RND*, respectively. Across all generated task systems,* FINE-RND *was at least (resp., at most) 0.13, 0.002, 0.35, 0.002, and 0.002 (resp., 1.0, 5.8, 3.5, 5.8, and 5.8) times* COARSE-RND*,* LIU-NO*,* WAN-UNR*,* YAN-UNR*, and* AHM-RND*, respectively.*

This can be seen in Fig. 8(a)–(c). On average, FINE-RND was significantly smaller than the decomposition-based approaches LIU-NO, YAN-UNR, and AHM-RND. The non-decomposition-based approach WAN-UNR was relatively closer to FINE-RND. Among all, COARSE-RND was on average the closest to FINE-RND. Note that, on average, non-decomposition-based analyses yielded smaller response-time bounds than decomposition-based analyses.

Among all generated task systems, there were cases in which FINE-RND was two orders of magnitude smaller than LIU-NO, YAN-UNR, and AHM-RND. However, there were also cases in which FINE-RND was larger than LIU-NO, YAN-UNR, and AHM-RND. This occurred when each node's response time was smaller than the DAG period under decomposition-based approaches. Furthermore, there were some systems in which FINE-RND was significantly smaller than COARSE-RND and WAN-UNR.

As seen in Fig. 8(a)–(c), for small processor counts, normalized utilizations, and edge-generation probabilities, the difference between WAN-UNR and FINE-RND was small, since WAN-UNR becomes tighter in such cases. As seen in Fig. 8(c), for large edge-generation probabilities, the difference between FINE-RND and LIU-NO (and YAN-UNR) was substantial. This occurred because each DAG was almost sequential, causing the maximum number of nodes from the source to the sink to equal the total number of nodes n. In such cases, LIU-NO and WAN-UNR became at least as large as $n \cdot T$, whereas FINE-RND and COARSE-RND were not larger than $vol(G) \leq mT$.

VIII. RELATED WORK

Most prior work on DAG-based task systems focused on constrained-deadline HRT systems [18], [28], [32], [33], [35], [44]. Graham gave an envelope-path-based analysis for a DAG task under any work-conserving scheduler [13]. More recently, multi-path bounds have been proposed that improve upon Graham's bound by considering multiple paths of the DAG [16]. Some work has also considered assigning node priorities to further refine Graham's bound [9], [17], [18], [44]. Other work on constrained-deadline HRT DAG tasks has provided schedulability tests by deriving *speed-up factors* or *resource-augmentation bounds* [5], [8], [24]. Decomposition-based analysis for constrained-deadline DAG tasks has also been studied under GEDF and fixed-priority schedulers [20], [32], [35]. These bounds can be applied to multi-DAG systems using *federated scheduling* techniques [6], [23] or by accounting for inter-DAG interference [18], [24], [28], [31]. For all work on constrained-deadline HRT DAGs, node-level self-dependencies are implicitly satisfied, obviating the need to consider them in response-time analysis. Real-time scheduling of constrained-deadline HRT DAG tasks has also been studied under heterogeneous multiprocessor platforms [2], [15], [19], [36].

For arbitrary-deadline HRT systems, most work assumes no node-level self-dependencies (*i.e.*, unrestricted parallelism) [14], [40], [42], [43]. Schedulability tests for multiple DAG tasks under GEDF and fixed-priority schedulers have been derived by determining speed-up factors [7], [8]. Other work has determined response-time bounds for multiple DAG tasks by considering envelope paths [30], [39], [42]. Fonseca *et al.* developed an envelope-path-based approach to determine response-time bounds for multiple arbitrary-deadline DAG tasks [12]. However, this approach requires that a DAG job does not commence execution before all previous DAG jobs finish, which can lead to utilization loss.

Decomposition-based analysis for systems with multiple DAG tasks with $P_i = 1$ values was first provided by Liu and Anderson [25]. Such analyses have also been developed for systems with unrestricted and arbitrary degrees of parallelism [1], [4], [43]. All of this work achieves soft-real-time optimality by ensuring no utilization loss. However, no non-decomposition-based analysis that guarantees SRT-optimality is currently known. Exact conditions for SRT-feasibility of sporadic task systems with arbitrary degrees of parallelism on a heterogeneous multicore machine were given in [27].

IX. CONCLUSION

We have presented an SRT-optimal scheduling policy for DAG tasks that cause no utilization loss for a single DAG task and enables non-decomposition-based analysis. Our policy can handle systems that have node-level self-dependencies. The scheduling algorithm can also be applied to arbitrary-deadline HRT DAG tasks. The presented experiments showed that our analysis enables response-time bounds to decrease by an order of magnitude on average.

In future work, we intend to devise SRT-optimal scheduling policies and corresponding non-decomposition-based analyses for multiple DAG tasks. We also plan to consider systems with non-preemptive nodes. Additionally, we aim to investigate synchronization policies for sharing resources among processing graphs scheduled by our proposed scheduler.

[6] Any schedule assuming no parallelism is also a valid schedule under unrestricted parallelism. Thus, any bound assuming no parallelism also holds when unrestricted parallelism is allowed.

REFERENCES

[1] S. Ahmed and J. Anderson, "Exact response-time bounds of periodic DAG tasks under server-based global scheduling," in *RTSS'22*, 2022, pp. 447–459.

[2] S. Ahmed, D. Massey, and J. Anderson, "Scheduling processing graphs of gang tasks on heterogeneous platforms," in *RTAS'25*, 2025, pp. 362–374.

[3] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "An empirical survey-based study into industry practice in real-time systems," in *RTSS'20*, 2020, pp. 3–11.

[4] T. Amert, S. Voronov, and J. Anderson, "OpenVX and real-time certification: The troublesome history," in *RTSS'19*, 2019, pp. 312–325.

[5] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *ECRTS'14*, 2014, pp. 97–105.

[6] ——, "Federated scheduling of sporadic DAG task systems," in *IPDPS'15*, 2015, pp. 179–186.

[7] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *RTSS'12*, 2012, pp. 63–72.

[8] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *ECRTS'13*, 2013, pp. 225–233.

[9] S. Chang, R. Bi, J. Sun, W. Liu, Q. Yu, Q. Deng, and Z. Gu, "Toward Minimum WCRT Bound for DAG Tasks Under Prioritized List Scheduling Algorithms," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 3874–3885, 2022.

[10] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *SIMUTools'10*, 2010, p. 60.

[11] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *WATERS'10*, 2010, pp. 6–11.

[12] J. C. Fonseca, G. Nelissen, and V. Nélis, "Schedulability analysis of DAG tasks with arbitrary deadlines under global fixed-priority scheduling," *Real-Time Syst.*, vol. 55, no. 2, pp. 387–432, 2019.

[13] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. of Appl. Math.*, vol. 17, no. 2, pp. 416–429, 1969.

[14] F. Guan, L. Peng, and J. Qiao, "A new federated scheduling algorithm for arbitrary-deadline DAG tasks," *IEEE Trans. Comput.*, vol. 72, no. 8, pp. 2264–2277, 2023.

[15] M. Han, N. Guan, J. Sun, Q. He, Q. Deng, and W. Liu, "Response Time Bounds for Typed DAG Parallel Tasks on Heterogeneous Multi-Cores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2567–2581, 2019.

[16] Q. He, N. Guan, M. Lv, X. Jiang, and W. Chang, "Bounding the response time of DAG tasks using long paths," in *RTSS'22*, 2022, pp. 474–486.

[17] Q. He, X. Jiang, N. Guan, and Z. Guo, "Intra-Task Priority Assignment in Real-Time Scheduling of DAG Tasks on Multi-Cores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2283–2295, 2019.

[18] Q. He, M. Lv, and N. Guan, "Response time bounds for DAG tasks with arbitrary intra-task priority assignment," in *ECRTS'21*, vol. 196, 2021, pp. 8:1–8:21.

[19] Q. He, Y. Sun, M. Lv, and W. Liu, "Efficient Response Time Bound for Typed DAG Tasks," in *RTCSA'23*, 2023, pp. 226–231.

[20] X. Jiang, X. Long, N. Guan, and H. Wan, "On the decomposition-based global EDF scheduling of parallel real-time tasks," in *RTSS'16*, 2016, pp. 237–246.

[21] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: enabling autonomous vehicles with embedded systems," in *ICCPS'18*, 2018, pp. 287–296.

[22] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *WATERS'15*, 2015.

[23] J. Li, K. Agrawal, C. Gill, and C. Lu, "Federated scheduling for stochastic parallel real-time tasks," in *RTCSA'14*, 2014, pp. 1–10.

[24] J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of global EDF for parallel tasks," in *ECRTS'13*, 2013, pp. 3–13.

[25] C. Liu and J. Anderson, "Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss," in *RTSS'10*, 2010, pp. 3–13.

[26] ——, "Supporting graph-based real-time applications in distributed systems," in *RTCSA'11*, 2011, pp. 143–152.

[27] D. Massey, S. Ahmed, and J. Anderson, "On the feasibility of sporadic tasks with restricted parallelism on heterogeneous multiprocessors," in *RTNS'24*, 2024, pp. 105–116.

[28] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *ECRTS'15*, 2015, pp. 211–221.

[29] M. Nasri, G. Nelissen, and B. B. Brandenburg, "Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling," in *ECRTS'19*, 2019, pp. 21:1–21:23.

[30] A. Parri, A. Biondi, and M. Marinoni, "Response time analysis for G-EDF and G-DM scheduling of sporadic DAG-tasks with arbitrary deadline," in *RTNS'15*, 2015, pp. 205–214.

[31] R. Pathan, P. Voudouris, and P. Stenström, "Scheduling parallel real-time recurrent tasks on multicore platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 915–928, 2018.

[32] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global EDF scheduling of directed acyclic graphs on multiprocessor systems," in *RTNS'13*, 2013, pp. 287–296.

[33] M. Qamhieh, L. George, and S. Midonnet, "A stretching algorithm for parallel real-time DAG tasks on multiprocessor systems," in *RTNS'14*, 2014, p. 13.

[34] R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors," in *ICDCS'90*, 1990, pp. 116–123.

[35] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Parallel real-time scheduling of DAGs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3242–3252, 2014.

[36] M. A. Serrano and E. Quiñones, "Response-time analysis of DAG tasks supporting heterogeneous computing," in *DAC'18*, 2018, pp. 125:1–125:6.

[37] J. Sun, N. Guan, J. Sun, X. Zhang, Y. Chi, and F. Li, "Algorithms for computing the WCRT bound of OpenMP task systems with conditional branches," *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 57–71, 2021.

[38] Z. Tong, S. Ahmed, and J. Anderson, "Holistically Budgeting Processing Graphs," in *RTSS'23*, 2023, pp. 27–39.

[39] N. Ueter, M. Günzel, G. von der Brüggen, and J. Chen, "Parallel path progression DAG scheduling," *IEEE Trans. Comput.*, vol. 72, no. 10, pp. 3002–3016, 2023.

[40] N. Ueter, G. von der Brüggen, J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *RTSS'18*, 2018, pp. 482–494.

[41] S. Voronov, S. Tang, T. Amert, and J. H. Anderson, "AI meets real-time: Addressing real-world complexities in graph response-time analysis," in *RTSS'21*, 2021, pp. 82–96.

[42] K. Wang, X. Jiang, N. Guan, D. Liu, W. Liu, and Q. Deng, "Real-time scheduling of DAG tasks with arbitrary deadlines," *ACM Trans. Design Autom. Electr. Syst.*, vol. 24, no. 6, pp. 66:1–66:22, 2019.

[43] K. Yang, M. Yang, and J. H. Anderson, "Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms," in *RTNS'16*, 2016, pp. 349–358.

[44] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *RTSS'20*, 2020, pp. 128–140.