White Rose Research Online

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of York.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/201373/

Version: Accepted Version

Proceedings Paper:
Chen, Nan, Zhao, Shuai, Gray, Ian et al. (2023) Precise Response Time Analysis for Multiple DAG Tasks with Intra-task Priority Assignment. In: Proc. 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE Real-Time and Embedded Technology and Applications Symposium. IEEE, pp. 174-184.

https://doi.org/10.1109/RTAS58335.2023.00021

Reuse
This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Precise Response Time Analysis for Multiple DAG Tasks with Intra-task Priority Assignment

Nan Chen*, Shuai Zhao†, Ian Gray*, Alan Burns*, Siyuan Ji*, Wanli Chang‡§
* Department of Computer Science, University of York, UK
† Department of Computer Science, Sun Yat-Sen University, China
‡College of Computer Science and Electronic Engineering, Hunan University, China
§Central Software Institute, Huawei Technologies, China

***Abstract*—In many real-time application domains, there are execution dependencies, such tasks may be formulated as multiple Directed Acyclic Graphs (DAGs) and scheduled with intra-task (i.e., intra-DAG) priority assignment. The worst-case completion time of a DAG must be bounded and schedulability analysis must be conducted during the design phase to estimate the required hardware resources. Typical examples include automotive systems and Ultra-Reliable Low Latency Communications (URLLC), which is the "to-business" protocol in 5G technologies, deployed in industrial automation for instance. To bound the execution time of multiple DAGs, there are two key factors to analyze: the intra-task interference for a single DAG and the inter-task interference between DAGs. While extensive efforts have been invested, the existing methods either still contain a large degree of pessimism or are even erroneous due to errors in the derived analysis. In this paper, we first provide an in-depth analysis of the limitation and defects of the existing methods. Inspired by these observations, we construct novel response time analysis for multiple DAG tasks with arbitrary intra-task priority assignment. Our analysis precisely accounts for both the intra- and inter-task interference by fully exploring the node parallelism in each DAG as well as between DAGs. Extensive experimental results show that the proposed analysis obtains tighter bounds and improves the system scheduability by at least 300% compared to state-of-the-art approaches. This improvement is even larger when the scheduling pressure is relatively high, up to 100% versus 0% in many cases. This work notably advances the use of response time analysis in the industry. Practitioners have to resort to either potentially unsafe measurement results or significant resource over-provisioning when precise analysis is unavailable.**

I. Introduction

With emerging real-time application scenarios, such as in autonomous systems and Ultra-Reliable Low Latency Communications (URLLC) in the industrial automation domain, complex functionalities are increasingly deployed in multi-core real-time systems. In these systems, the functionalities can be delivered in a concurrent fashion, but are subject to execution dependencies at certain points of execution [1]. To reflect the complex dependency and parallelism that widely exist between computations in the system, the Directed Acyclic Graph (DAG) task model has received much attention in the real-time systems community [2].

A DAG task contains a set of nodes and directed edges, in which each node indicates a code segment that must be executed in a sequential manner and a directed edge connecting two nodes indicates their execution dependency [2]. Any two nodes that have no execution dependencies between each other can be executed in parallel across multiple cores. Compared to the traditional sporadic task model [3] in which tasks have no execution dependencies between each other, the DAG task abstraction provides a more realistic model that better describes the internal execution relationships within the system. However, it also imposes new research challenges in ensuring system schedulability as the traditional bound on the worst-case completion time of regular task models (e.g. sporadic and periodic) does not take these existing execution dependencies within DAGs into account.

Research into the response time analysis of DAGs covers a wide range of scheduling schemes, including global [4], [5], fully-partitioned [6], and federated [7]. Focusing on a common setup, where multiple DAGs are running on a symmetric multi-core system with a global scheme, extensive research efforts have been conducted to provide safe and accurate analytical bounds on the key factor of the completion time of tasks: task interference. The interference is defined as the behavior of a task when it runs on a processor and prevents another ready task from execution [8]. With DAG tasks, the interference is categorized as either intra-task or inter-task interference [4]. The intra-task interference indicates the node-level interference within a DAG task whilst the inter-task interference means the delay caused by other DAGs.

In general, the existing analysis techniques can be categorized into two types: the *generic bound* [4], [9], [10] and the *priority-explicit bound* [11]–[14]. For the generic bound, Melani et al. [4], Fonseca et al. [9] and Serrano et al. [10] provide analytical bounds for DAG tasks under various scheduling methods, e.g., preemptive [4] and limited preemption [10]. Such analysis provides a generic bound on the intra-task interference for all possible node priority ordering under any work-conserving schedule [4]. However, it has been shown in [11] and [13] that this bound can be pessimistic if the intra-task priority assignment of DAG tasks is known before the execution of the system.

With the knowledge of the intra-task priorities, He et al. [11], [12] presented new analysis techniques that only take the high-priority nodes into account when bounding the intra-task interference; they obtained more accurate analytical results when compared to the generic bound. However, the above analysis still has a large degree of pessimism, especially

when bounding the inter-task interference. This is because the potential parallelism of the DAGs is not fully considered by the analysis, in which the workload that is actually executed in parallel can be accounted for in the interference [13]. Zhao et al. [13], [14] presented an analysis that attempts to explicitly compute the parallel workload to obtain a tighter bound on the intra-task interference. However, as shown in this paper, such computations have resulted in a highly-complicated and error-prone analysis due to the complex nature of DAGs, and are hard to apply in multi-DAG work-conserving systems as the inter-task interference cannot be computed using the same approach.

Therefore, with the existing DAG analyzing methods, the system engineers need to either resort to potentially unsafe timing bounds or provide resource over-provisioning. This has imposed a significant barrier to the application of the response time analysis of DAG tasks in real-world systems.

Main contributions: This paper focuses on periodic DAG tasks running on a symmetric multi-core systems with a global limited preemption scheme. The principal contribution of this paper is a novel priority-explicit response time analysis that advances the industrial usage of response time analysis for DAGs in real-world systems. The proposed analysis avoids the highly complicated computations while producing tighter upper bounds for DAG tasks by precisely bounding the intra-task and inter-task interference of a DAG task. To achieve this, the following detailed contributions are presented in this paper:

- An in-depth analysis and illustration of the limitations and defects of the existing response time analysis for DAGs.
- New analysis techniques for a single DAG that fully explores the node-level parallelism and safely removes the workload that cannot cause a delay when bounding the intra-task interference.
- A new analysis method for multi-DAGs that precisely identifies the nodes of other DAGs that can impose a delay on the current DAG by computing the inter-task interference.

Extensive experiments demonstrate that compared to the state-of-the-art methods in [10] and [12], the constructed analysis achieves tighter bounds and can improve the system schedulability by over 300%, and has an even larger advantage for systems with high schedulability pressure, i.e., can schedule up to 100% of such systems against 0% for the competing methods.

Paper organization: Section II presents the task and scheduling models assumed in this paper. Section III describes the related work on techniques for DAG tasks. Section IV then provides a disccussion on the limitations of existing analysis. Motivated by these limitations, Section V constructs a novel response time analysis for DAG tasks and Section VI presents experimental results. Finally, Section VII draws some conclusions.

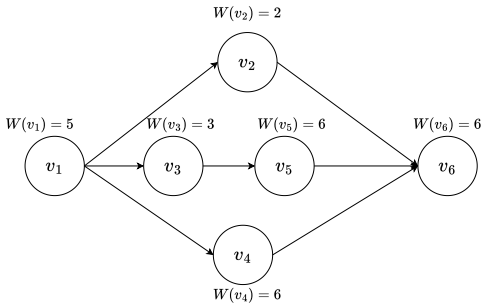

Fig. 1: A DAG task.

II. System Model

A. Task model

A real-time DAG task τ_i is defined by $\{T_i, D_i, \mathcal{G}_i = (V_i, E_i), P_{\tau_i}\}$, in which T_i denotes its period (or the minimum inter-arrival time as the worst-case in a sporadic release model [3]), D_i gives a constrained relative deadline, i.e., $D_i \leq T_i$, and $\mathcal{G}_i$ is a graph defining the set of activities forming the task. The graph is defined as $\mathcal{G}_i = (V_i, E_i)$ where V_i denotes the set of nodes and $E_i \subseteq (V_i \times V_i)$ gives the set of directed edges connecting any two nodes. A node in DAG τ_i is denoted as $v_{i,j} \in V_i$, where the index j gives the node index and index i means it belongs to τ_i. To ease presentation, the subscript of the DAG task (i.e., i for τ_i) is omitted when we consider a single DAG task. Each task τ_i and node v_j are assigned an individual priority denoted as P_{τ_i} and P_{v_j} respectively. In this paper, we assume that the larger the number the higher the priority. Moreover, we assume that if the priority of τ_i is larger than τ_j (i.e. $P_{\tau_i} > P_{\tau_j}$), then all of the nodes in τ_i have a higher priority than nodes in τ_j (i.e. $P_{v_a} > P_{v_b}, \forall v_a \in V_i$ and $v_b \in V_j$). The notation W is applied to represent the total Worst-Case Execution Time (WCET) of a node or a node list. For example, the WCET of a node v_j is denoted as $W(v_j)$ and the total WCET of a DAG τ_i is denoted as $W(V_i) = \sum_{v_j \in V_i} W(v_j)$.

If two nodes v_j and v_k are connected by a directed edge (i.e., $(v_j, v_k) \in E$), v_k can start executing only if v_j finishes executing. That is, v_j is a *predecessor* of v_k, whereas v_k is a *successor* of v_j. The predecessors and successors of node v_j can be formally defined as $pre(v_j) = \{v_k \in V_i \mid (v_k, v_j) \in E\}$ and $suc(v_j) = \{v_k \in V_i \mid (v_j, v_k) \in E\}$, respectively. Nodes that are either *direct* or *transitive* predecessors and successors of a node v_j are termed as its ancestors $anc(v_j)$ and descendants $des(v_j)$ respectively. A node v_j with $pred(v_j) = \varnothing$ or $succ(v_j) = \varnothing$ is referred to as the *source* v_{src} or *sink* v_{sink} respectively. As with most of the existing work [11], [12], [14], we assume a DAG task always starts with one source and ends with one sink node, otherwise, a dummy node without utilization is added to the beginning or the end of the DAG.

Based on the DAG task model, we define the features of a DAG that will be utilized by the response time analysis constructed in this paper.

Definition 1. *(Concurrent nodes) For a pair of nodes $v_j, v_k \in V_i$, if v_j is neither the ancestors of v_k (i.e. $v_j \notin anc(v_k)$) nor the descendants of v_k (i.e. $v_j \notin des(v_k)$), it is referred to as a concurrent node of v_k. The set of concurrent nodes of v_j is denoted as $C(v_j)$ and is expressed as Equation 1.*

$$C(v_j) = \{v_k | v_k \in V_i \wedge \\ v_k \notin anc(v_j) \wedge v_k \notin des(v_j) \wedge v_k \neq v_j\} \quad (1)$$

Example 1. *Taking the DAG task shown in Figure 1 as an example. The concurrent nodes of v_2 within the DAG is $C(v_2) = \{v_3, v_4, v_5\}$.*

With the definition of concurrent nodes, we also provide the definition for *Interference* which is a key factor in the response time analysis of real-time systems.

Definition 2. *(Interference) the amount of time spent executing high-priority tasks when a low-priority task is runnable [3].*

When a low-priority task executes non-preemptively, it can also prevent a high-priority task from executing; this is referred to as the *low-priority interference* in this paper – in other works, it is also referred to as blocking [13].

With the DAG task model, the *Interference* can be further categorized as *intra-task* and *inter-task* interference. They can be defined as follows.

Definition 3. *(Intra-task interference) the amount of interference incurred by a node from nodes within the same DAG.*

Definition 4. *(Inter-task interference) the amount of interference incurred by a node from nodes from other DAGs.*

A complete path within a DAG is defined as follows. As no partial paths will be used in this paper, we use the term path directly.

Definition 5. *(Path) A path of τ_i is denoted by $\lambda_a = \{v_0, \cdots, v_k\}$, where $\forall p \in [0, k], (v_p, v_{p+1}) \in E$, $v_0 = v_{src}$ and $v_k = v_{sink}$.*

For a path λ_a, $W(\lambda_a)$ returns its total workload (i.e., length), which is the sum of the WCET of nodes in λ_a. The path with the highest workload ($W(\lambda_a)$) within a DAG τ_i is defined as the critical path and is denoted as λ_i^*.

Example 2. *As shown in Figure 1, the DAG task consists of three paths:$\lambda_1 = \{v_1, v_2, v_6\}$, $\lambda_2 = \{v_1, v_3, v_5, v_6\}$ and $\lambda_3 = \{v_1, v_4, v_6\}$, with $W(\lambda_1) = 5 + 2 + 6 = 13$, $W(\lambda_2) = 5 + 3 + 6 + 6 = 20$ and $W(\lambda_3) = 5 + 6 + 6 = 17$, respectively Therefore, λ_2 is the critical path of this DAG task.*

Following the single-DAG task model, a multi-DAG system contains n recurrent DAG tasks $\Gamma = \{\tau_1, ..., \tau_n\}$, in which each task τ_i is assigned a unique priority $P(\tau_i)$. As with [9], [11], we claim that the DAG tasks in this work are independent of each other, that is we assume nodes of two different DAGs do not share any resources and there exists no dependency between them.

B. System and scheduling model

In this work, we consider a symmetric multi-core system with m cores and apply a global fixed-priority scheduling method with the limited preemption scheme [10]. That is, a late-arriving high-priority node can be blocked by low-priority nodes that are currently executing, i.e., non-preemptive at the node level. However, a newly-arrived high-priority DAG task can preempt an executing low-priority DAG on a core when the currently executing node has finished, i.e., preemption is allowed between the execution of two tasks.

For all the DAG tasks that are ready to execute, the scheduler follows the rule of the *highest priority DAG first*, and within a DAG task, it follows the *highest priority node first* execution order. That is, task priority is used to select the next task to execute in the ready queue, whereas node priority gives the execution order of nodes in the scheduled DAG. We assume that a higher numeric value indicates a higher priority. However, we note that the response time analysis constructed in this paper does not rely on any specific intra-task and inter-task priority assignment, i.e., it is compatible with any given priority order at both the DAG and the node level.

III. Related Work

There exists a large body of work on scheduling and analyzing DAG tasks, which covers a wide range of hardware settings, scheduling schemes and task models [4], [9]–[11], [13]–[18]. In this section, we focus on the major existing analysis for DAG tasks with a global work-conserving schedule in symmetric multi-core systems. In Section IV, an in-depth analysis is provided to illustrate the major limitations and defects of the existing analysis.

A. Generic DAG analysis under work-conserving schedules

The majority of existing methods on DAG scheduling assume a work-conserving scheduling scheme [4]. For this schedule, Graham [19] proposes the classic generic bound for the traditional tasks, in which the interference of a task is computed by the average value of the total workload of other tasks on all processors.

Based on the classic bound, Melani et al. [4] construct an analysis to compute the worst-case makespan of DAG tasks under a preemptive scheduling scheme. This analysis takes the priorities of the DAG tasks as an input but assumes no knowledge of the intra-node priority of each DAG. Under this analysis, two key factors of the worst-case makespan of a DAG task are identified: the intra-task and inter-task interference, in which they are computed using a similar approach to that in [19]. We note that the analysis is constructed for conditional DAGs, i.e., a DAG task can contain conditional edges that may not be active during execution, but is fully compatible with the unconditional ones. Later on, this analysis is improved by Fonseca et al. [9] using a finer-grained approach for computing the inter-task interference caused by carry-in and carry-out DAGs. Following a similar approach, Serrano et al. [10] construct a generic analysis for DAG tasks for the limited

preemption scheme, where a blocking factor imposed by the low-priority nodes and DAGs is introduced in the analysis.

As described in Seciton I, the generic bound is useful when an exact node execution order is not known before the system execution. However, if such information is obtained, this bound can become significantly pessimistic due to the over-calculated interference (see in Limitations I and II in Section IV) [13].

B. DAG analysis with explicit intra-task priority

With an explicit intra-node priority ordering, He et al. [11] present a new response time analysis for DAG tasks under the preemptive scheme. This analysis computes the makespan of a DAG task by 1) examining the worst-case finish time of each path and 2) only taking the concurrent nodes with a higher priority as the intra-task interference. By doing so, it provides a tighter bound on makespan and dominates the generic bound in [4]. In [12], an improved analysis is constructed which can be applied to any node ordering rules along with a new intra-task priority assignment that further increases the schedulability of the system. However, as the same with the traditional bound [4], both the intra-task and inter-task interference in [11] and [12] are computed using the average value of the interfering workload on all processors. This approach contains a certain degree of pessimism as it does not explicitly consider the parallel workload that cannot cause a delay to the current DAG task (see Limitations II and III). In particular, such pessimism can become significant when accounting for the inter-task interference, and hence, greatly undermines system schedulability.

In [13], [14], Zhao et al. presented a new response time analysis of a DAG task under the limited preemption scheme. By examining the internal structure of the DAG and the node ordering, this analysis explicitly accounts for the parallel workload that is executed with the longest path of the DAG, and subtracts such workload when accounting for the intra-task interference. From the results, this analysis provides a tighter bound than [11] due to the awareness of the parallel workload in a DAG. However, the analysis approaches adopted in [13] have led to over-complicated and error-prone computations due to the highly-complex nature of a DAG task (see Lemma 3 in Section IV). In addition, this analysis does not support multi-DAGs under the work-conserving schedule as the inter-task interference cannot be computed using the same approach.

IV. Limitations of state-of-the-art

The first classical response time analysis of DAG scheduling under the preemptive approach is provided in [4]. Later on, the analysis is further extended to the limited preemption schedule with the inclusion of interference from low-priority nodes in [10]. In this paper, we focus on systems with limited preemption, we first briefly introduce the classical bound for limited preemption scheduling scheme in [10] and present its limitations. Then, we will analyze the limitations that have been alleviated by the state-of-the-art methods and ones that can still be improved.

$$R_i = \underbrace{Makespan}_{\text{part 1}} + \underbrace{Interference}_{\textbf{part 2}} \tag{2}$$

$$R_i = \underbrace{W(\lambda_i^*) + \frac{(W(V_i) - W(\lambda_i^*)}{)}\, m_{\text{part 1}}}_{} + \underbrace{\left\lfloor \frac{I_i^{lo} + I_i^{hi}}{m} \right\rfloor}_{\textbf{part 2}} \tag{3}$$

As shown in Equation 2, the worst-case response time analysis of [10] can be divided into two parts: 1) the worst-case makespan of a DAG, and 2) the inter-task interference suffered from other DAG tasks. As shown in Equation 3, the first part calculates the worst-case makespan of a DAG task τ_i. The $W(\lambda_i^*)$ term is the sum of the WCET of the nodes in the critical path. The analysis includes the principle that all nodes apart from the critical path can provide certain intra-task interference. Therefore, the amount of intra-task interference incurred by the critical path is denoted as $\frac{1}{m}(W(V_i) - W(\lambda_i^*))$, in which $W(V_i)$ gives the total workload of τ_i and $W(V_i) - W(\lambda_i^*)$ represents the amount of workload of a DAG apart from the critical section. The workload after subtraction is then divided by the number of cores in the system m to work out the upper bound of the interference delay.

Then, the second part calculates the inter-task interference of τ_i, in which I_i^{hi} is the workload of the interference imposed by high-priority tasks, which is the sum of the WCETs of all high-priority nodes that are released during the execution of τ_i. As for I_i^{lo}, it contains the workload of the interference from low-priority tasks due to limited preemption scheduling. To bound I_i^{lo}, Lemma 1 and 2 are constructed in [20].

Lemma 1. *For a DAG task, the first node can suffer a delay from at most m (total core number) nodes with lower priorities [20].*

Lemma 2. *For a DAG task, nodes apart from the first node can suffer the interference delay from at most $m-1$ nodes with lower priorities [20].*

According to Lemma 1 and 2, the classical bound then assumes the first node (resp. every node apart from the first node) can suffer delay from m (resp. $m-1$) largest low-priority nodes that can execute in parallel [10]. The largest m (and $m-1$) nodes among the DAG tasks with lower priorities that can execute in parallel are denoted as δ_i^m (and δ_i^{m-1}). As shown in Equation 4, I_i^{lo} is equal to the total workload of δ_i^m (incurred by the first node) plus $||V_i||-1$ times the workload of δ_i^{m-1} (incurred by rest of the nodes), where $||V_i||$ represents the size of the nodes in τ_i

$$I_i^{lo} = W(\delta_i^m) + (||V_i||-1) * W(\delta_i^{m-1}); \tag{4}$$

An example of the calculation of the classical bound [10] is presented below.

Example 3. *We first calculate the worst-case makespan of τ_1 shown in Figure 2, which is the first part of Equation 3. The*

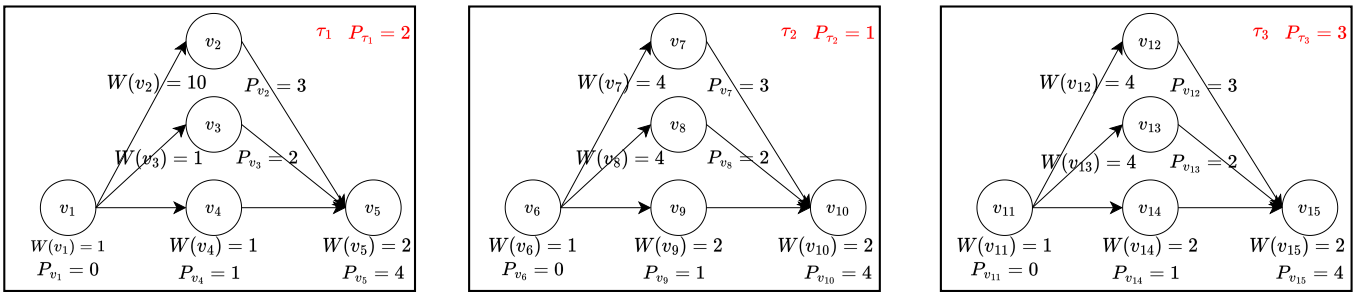

Fig. 2: Three DAG tasks with different priorities.

critical path of τ_1 *is* $\lambda_1^* = \{v_1, v_2, v_5\}$. *The total workload of the* λ_1^* *is* $W(\lambda_1^*) = 1+10+2 = 13$. *The total workload of* τ_1 *is* $W(V_1) = 1+10+1+1+2 = 15$. *For a dual-core system, the interference of* λ_1^* *can be calculated as* $\frac{15-13}{m} = \frac{15-13}{2} = 1$. *The worst-case makespan is equal to* $R_1 = W(\lambda_1^*) + 1 = 13+1 = 14$.

Then, we calculate the inter-task interference. For simplicity, assuming τ_2 *and* τ_3 *will only release once during the execution of* τ_1. *Task* τ_3 *has higher priority than* τ_1, *i.e* $P_{\tau_3} = 3 > P_{\tau_1} = 2$, *hence* $I_1^{hi} = W(V_3) = 1+4+4+2+2 = 13$. *Task* τ_2 *has lower priority than* τ_1, *We have* $\delta_1^2 = \{v_7, v_8\}$ *and* $\delta_1^1 = \{v_7\}$, *then* $W(\delta_1^2) = 4+4 = 8$ *and* $W(\delta_1^1) = 4$, *the size of* τ_1 *the* $||V_1|| = 5$. *Finally,* $I_1^{lo} = 8 + (5-1)*4 = 24$. *According to Equation 3, we have* $R_1 = 14 + \lfloor \frac{13+24}{2} \rfloor = 32$

Based on the description provided above, we can see that the classical bound for systems with limited preemptions proposed in [10] is rather pessimistic. The main limitations of the classical bound can be concluded as follows.

Limitation I: The analysis neglects the parallelism feature between nodes and results in a pessimistic calculation of the worst-case response time.

For example, for the analysis of a single DAG, the classical bound assumes all the nodes apart from the critical path will interfere with the critical path. However, some nodes execute in parallel without delaying the execution of the critical path. Moreover, when accounting for the inter-task interference, it also assumes all nodes with a higher priority will interfere with the DAG. For a system with sufficient cores available, DAG tasks can also execute in parallel without interfering with each other.

Limitation II: When calculating the inter-task low-priority inference, each node apart from the source node is assumed to incur a low-priority interference from δ_i^{m-1}, which is pessimistic.

According to Lemma 2, each node apart from the source node will suffer interference from at most $m-1$ low-priority nodes. However, each low-priority node can only delay a high-priority node once during one release [3]. Thus, assuming nodes in δ_i^{m-1} can interfere with all nodes apart from the first node will result in repetitive calculations of interference.

He et al. [12] alleviates Limitation I of the classical bound by introducing priorities to nodes within a DAG. They consider systems with a preemptive scheduler. With a given intra-task priority assignment, only nodes with higher priority need to be considered as the interference workload. Therefore, the resulting intra-task interference can be reduced. As shown in the Equation 5, $W(\lambda_j)$ calculates the workload of the path λ_j. The factor $I(\lambda_i)$ denotes the set of nodes that have a higher priority than the nodes in path λ_i. Term $W(I(\lambda_i))$ then calculates the interference workload of the path. Finally, $max_{\lambda_j \in \tau_i}\{\}$ iterates through all paths in τ_i and finds out the maximum bound on the DAG makespan. In this analysis, the intra-task interference is reduced by assuming an intra-task priority assignment under the preemptive scheme. However, the parallelism between nodes is not fully exploited. For example, the analysis assumes that all nodes with a higher priority contribute intra-task interference when computing the worst-case makespan. Yet, when there are sufficient cores in the system, nodes will execute in parallel and do not suffer from any interference. Hence, the analysis of the DAG makespan can be improved with respect to limitation I. Moreover, as it focuses on preemptive scheduling, it does not contribute to Limitation II.

$$R_i = \underbrace{\max_{\lambda_j \in \tau_i} \left\{ W(\lambda_j) + \frac{W(I(\lambda_j))}{m} \right\}}_{\text{part 1}} + \underbrace{\left\lfloor \frac{1}{m} I_i^{hi} \right\rfloor}_{\textbf{part 2}} \tag{5}$$

Later, Zhao et al. [14] proposed a response time bound under the limited preemption scheme which aims to eliminate unnecessary interference by exploring the parallelism of the execution of nodes. The analysis produces the worst-case makespan by computing the worst-case finish time of each node. According to Equations 3, 4, and 10 in [14], the key concept of calculating the worst-case finish time of each node can be summarized as Equation 6. In this analysis, the worst-case finish time of a node v_j is denoted as $F(v_j)$, which is computed by determining: 1) the WCET of (v_j), 2) the predecessor node with the latest worst-case finish time, i.e. $\max_{v_k \in pre(v_j)}\{F(v_k)\}$, and 3) the intra-task interference incurred by v_j (denoted as $\frac{W(C(v_i) \setminus \bigcup_{v_k \in anc(v_j)} I_{v_k})}{m}$).

$$\begin{aligned} F(v_j) = W(v_j) + & \max_{v_k \in pre(v_j)} \{F(v_k)\} \\ & + \left\lceil \frac{W(C(v_j) \setminus \bigcup_{v_k \in anc(v_j)} I_{v_k})}{m} \right\rceil \end{aligned} \tag{6}$$

We note that the calculation of the intra-task interference utilizes $C(v_j)$ to exclude all the interference that has been

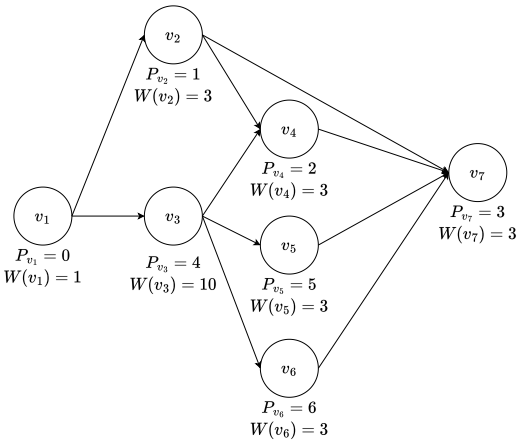

Fig. 3: An example of a DAG.

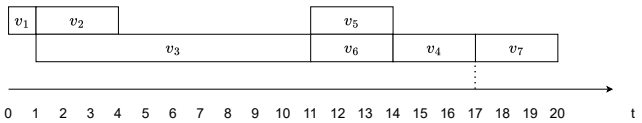

Fig. 4: The simulation graph of the DAG in Figure 3.

accounted for when computing the worst-case finish of the ancestors of v_j, in which function $W()$ returns the workload of the interference that is left. However, this analysis may not be safe due to Lemma 3.

Lemma 3. *The calculation of the worst-case finish time shown in Equation 6 is not necessarily the worst-case finish time.*

Proof. An example of a DAG task is shown in Figure 3. Assuming the DAG executes in a dual-core system. To ease the presentation, the scheduling scheme is assumed to be preemptive which will not affect our claim. We now compute the worst-case finish time of each node and the makespan of the DAG using the analysis in [14].

According to Equation 6, we can have $F(v_1) = W(v_1) = 1$ as it has no predecessors and concurrent nodes. As for v_2, the concurrent nodes are $C(v_2) = \{v_3, v_5, v_6\}$. These concurrent nodes have a higher priority than v_2 and the only ancestor node of v_2 (i.e., v_1) does not incur any inference. Accordingly, $F(v_2) = W(v_2) + F(v_1) + \frac{(W(v_3)+W(v_5)+W(v_6))}{2} = 3 + 1 + \frac{(10+3+3)}{2} = 12$ based on Equation 6. As for v_3, we have $F(v_3) = W(v_3) + F(v_1) = 10 + 1 = 11$, in which $C(v_3) = \{v_2\}$ with $P_{v_2} < P_{v_3}$. Similarly, $F(v_4) = W(v_4) + F(v_2) = 3 + 12 = 15$, in which $C(v_4) = \{v_5, v_6\}$, $pre(v_4) = \{v_2, v_3\}$, and $\max_{v_k \in pre(v_4)}\{F(v_k)\} = F(v_2) = 12$.

However, as shown in Figure 4, v_5 and v_6 will delay the execution of v_4, which results in a finish time of 17. However, such a delay is not accounted for in the analysis. Therefore, this analysis does not produce the worst-case finish time of the DAG. □

It is worth noting that, the analysis in [14] is very complicated and is impossible to thoroughly explain in this section. Here we only extract its main concept to validate our claim. As for the analysis of multi-DAGs, this analysis does not support work-conserving scheduling as the inter-task interference cannot be computed using the same analytical approach as for the intra-task interference. Therefore, as the bound in [14] may not be safe, Limitations I and II are still open to be resolved, which motivates our work in the following section.

V. Response Time Analysis

As described in Section IV, the existing bounds calculate the makespan of a single DAG by computing the worst-case finish time of each path, and then account for the worst-case response time in multi-DAGs by adding the inter-task interference upon the DAG makespan, which can result in a pessimistic bound. In this section, we construct a finer-grained response time analysis that bounds the worst-case makespan of a DAG as well as its worst-case response time in multi-DAGs by fully exploring the potential parallelism of each node in the system.

A. Worst-case makespan of a single periodic DAG

To analyze the worst-case makespan of a DAG task, we aim to identify the worst-case finish time of each node and then identify the node that finishes the latest in the DAG task. Equation 7 presents the worst-case makespan of a single DAG task, denoted as R[1]. Notation $F(v_j)$ defines the worst-case finish time of node v_j and $\max_{v_j \in V}\{F_{v_j}\}$ calculates the node that has the largest finishing time in the DAG.

$$R = \max_{v_j \in V}\{F(v_j)\} \tag{7}$$

Then for a given node v_j, its worst-case finish time is computed as the sum of its worst-case starting time $S(v_j)$ and its WCET $W(v_j)$, as shown in Equation 8.

$$F_{v_j} = S_{v_j} + W(v_j) \tag{8}$$

For a node v_j, it is allowed to start its execution if all of its predecessors have finished their executions. In addition, when v_j is eligible to execute, it can incur the intra-task interference imposed from the concurrent nodes with a higher priority in the DAG task. Equation 9 bounds the worst-case starting time of v_j, in which I_{v_j} gives the nodes that can cause an intra-task interference on node v_j. This equation iterates through each predecessor node $v_k \in pre(v_j)$, and computes the worst-case finish time of v_k and the associated intra-task interference that v_j can incur, and hence, derives a safe bound [11].

$$S_{v_j} = \max_{v_k \in pre(v_j)} \left\{ F(v_k) + \left\lceil \frac{W(I_{v_j} \setminus I_{v_k})}{m} \right\rceil \right\} \tag{9}$$

In addition, for the computation of the intra-task interference, we exclude the interference (i.e. $I_{v_j} \setminus I_{v_k}$) that has been accounted for in $F(v_k)$ to avoid repetitive calculations. The term $W(I_{v_j} \setminus I_{v_k})$ represents the total workload that could interfere v_j, and the final intra-task interference of v_j

[1]The index of the DAG task is committed in the single DAG case for the ease of presentation.

is given by $\lceil \frac{W(I_{v_j} \setminus I_{v_k})}{m} \rceil$. Below we present details on the identification of nodes in I_{v_j}.

Under the limited preemptive scheduling scheme with a defined priority assignment policy, a node v_j will not necessarily be delayed by all of its concurrent nodes $C(v_j)$. To bound the amount of delay suffered by v_j from $C(v_j)$, we first identify the nodes that can never delay v_j.

Lemma 4. *For a pair of concurrent nodes v_k and v_j with $P_{v_k} < P_{v_j}$, if v_k is eligible to execute (i.e., all of its predecessors have finish executing) at the same time as or later than v_j, v_k and its descendants will not impose a delay to v_j.*

Proof. If v_k is eligible to execute at the same time as or later than v_j, because $P(v_k) < P(v_j)$, once a core is available the scheduler will select v_j to execute first. Therefore, v_k and its descendants ($des(v_k)$) will not delay v_j. □

Lemma 5. *For a node v_j, nodes in the same DAG that are eligible to execute at the same time with or later than v_j are computed by $\eta(v_j)$, as shown in Equation 10.*

$$\eta(v_j) = \left\{ v_k | v_k \in V \wedge pre(v_j) \subseteq pre(v_k) \wedge v_j \neq v_k \right\} \quad (10)$$

Proof. For a node $v_k \in V$ and $v_j \neq v_k$, the predecessors of v_k ($pre(v_k)$) containing all the predecessors of v_j ($pre(v_j)$) can be expressed as $pre(v_j) \subseteq pre(v_k)$. Hence, the predecessors of v_k can finish later than or at the same time as the predecessors of v_j. Finally, we can deduce that v_k is eligible to execute at the same time with or later than v_j □

Based on Lemma 4 and 5, the set of nodes that cannot impose any delay to the node v_j (denoted as $I_{v_j}^{remove}$) can be identified in Equation 11. This equation iterates over all the nodes in the DAG and appends v_k and its descendants ($des(v_k)$) in $I_{v_j}^{remove}$, if the following conditions are satisfied: 1) the priority of v_k is smaller than v_j ($P_{v_k} < P_{v_j}$); and 2) $v_k \in \eta(v_j)$ or any of its ancestors belongs to $\eta(v_j)$ (i.e., $anc(v_k) \cap \eta(v_j) \neq \emptyset$).

$$I_{v_j}^{remove} = \bigcup_{v_k \in V} \begin{cases} v_k \cup des(v_k), & if\ P_{v_k} < P_{v_j} \wedge \\ & (v_k \in \eta(v_j) \\ & \vee anc(v_k) \cap \eta(v_j) \neq \emptyset) \\ \emptyset, & \text{otherwise.} \end{cases} \quad (11)$$

Theorem 1. *Equation 11 bounds the task set that will never delay the execution of a task.*

Proof. For a node v_k with priority $P_{v_k} < P_{v_j}$ that satisfies $v_k \in \eta(v_j)$, it cannot impose any delay to node v_j according to Lemma 4 and 5. Moreover, if any ancestors of v_k belong to $\eta(v_j)$ (i.e. $anc(v_k) \cap \eta(v_j) \neq \emptyset$), the ancestor is eligible to execute at the same time as or later than v_j according to Lemma 5. Accordingly, v_k is eligible to execute at the same time as or later than v_j, hence, v_k cannot impose any delay to node v_j according to Lemma 4. □

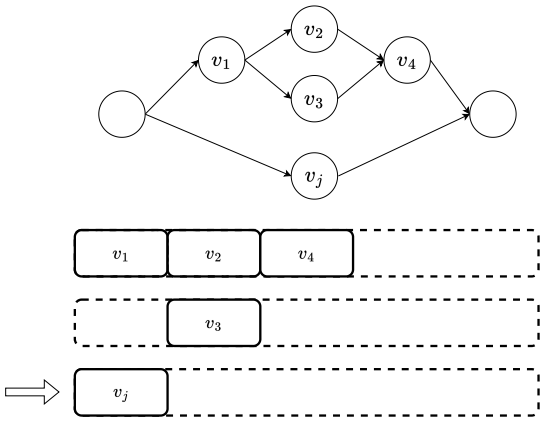

Fig. 5: An example of the interference-free execution.

Based on Theorem 1, Equation 12 bounds the set of nodes that can potentially cause intra-task interference with a node v_j, denoted as $I_{v_j}^{potential}$. As shown in the equation, $I_{v_j}^{potential}$ is computed by excluding the nodes that cannot interfere with v_j from the concurrent node of v_j (i.e. $C(v_j) \setminus I_{v_j}^{remove}$).

$$I_{v_j}^{potential} = C(v_j) \setminus I_{v_j}^{remove} \quad (12)$$

Lemma 6. *For a node v_j, if the interfering nodes in $I_{v_j}^{potential}$ cannot occupy all m cores in the system, then v_j will not suffer from the intra-task interference.*

Proof. Assuming all nodes in $I_{v_j}^{potential}$ execute on $m-1$ cores of a system, when v_j arrives, it will directly execute on the idle core without being delayed. □

Example 4. *As shown in Figure 5, the set of nodes that can potentially interfere with v_j is $I_{v_j}^{potential} = \{v_1, v_2, v_3, v_4\}$. Due the dependencies of nodes in $I_{v_j}^{potential}$, they can occupy at most two cores, therefore v_j can execute on the core available directly without incurring any interference delay.*

Let $path(I_{v_j}^{potential})$ denote a function that computes the largest number of cores that $I_{v_j}^{potential}$ can occupy. Based on Lemma 6, I_{v_j} can be computed by Equation 13. As shown in this equation, when $path(I_{v_j}^{potential}) < m$ (i.e., the potential interfering nodes cannot take all m cores), v_j does not incur any intra-task interference, and hence, I_{v_j} can be directly set to empty. Otherwise, v_j can incur a certain amount of intra-task interference from $I_{v_j}^{potential}$. As shown in Equation 13, under this case, the nodes that cause this interference can be categorized into the following three groups.

1) $I_{v_j}^{hi}$: the high-priority nodes in $I_{v_j}^{potential}$,
2) $I_{v_j}^{lomax}$: the $m-1$ largest nodes among the low-priority nodes in $I_{v_j}^{potential}$, and
3) $I_{v_j}^{lopre}$: the ancestor nodes of $I_{v_j}^{lomax}$ among the low-priority nodes in $I_{v_j}^{potential}$.

$$I_{v_j} = \begin{cases} \emptyset, & if\ path(I_{v_j}^{potential}) < m \\ I_{v_j}^{hi} \cup I_{v_j}^{lomax} \cup I_{v_j}^{lopre}, & \text{otherwise} \end{cases} \quad (13)$$

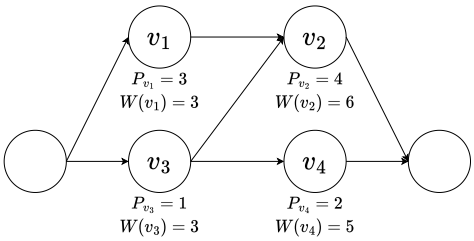

Fig. 6: An example of a DAG task.

Equation 14 provides the computation for $I_{v_j}^{hi}$, which takes all the high-priority nodes in $I_{v_j}^{potential}$ into account. In the worst case, when v_j arrives, all nodes in $I_{v_j}^{hi}$ have just started executing, therefore, $I_{v_j}^{hi}$ is included in I_{v_j}.

$$I_{v_j}^{hi} = \left\{ v_k | v_k \in I_{v_j}^{potential} \wedge P_{v_k} > P_{v_j} \right\} \tag{14}$$

Under the global limited preemption scheme, nodes in a DAG task can suffer a delay from at most $m-1$ low-priority nodes[2]. This is because before the node arrives, one core will be taken up by its predecessor and the rest $m-1$ cores can be taken by the concurrent low-priority tasks. Proofs of the above statements on the low-priority intra-task inference can be found in Lemmas 4.2 and 4.3 in [20]. Therefore, the low-priority nodes in $I_{v_j}^{potential}$ that can cause a delay to v_j can be computed by Equations 15 and 16, in which $I_{v_j}^{lo}$ returns all the low-priority nodes in $I_{v_j}^{potential}$ and $I_{v_j}^{lomax}$ identifies the $m-1$ nodes in $I_{v_j}^{lo}$ that can cause the maximum delay.

$$I_{v_j}^{lo} = \left\{ v_k | v_k \in I_{v_j}^{potential} \wedge P_{v_k} < P_{v_j} \right\} \tag{15}$$

$$I_{v_j}^{lomax} = \overset{m-1}{\max}\{I_{v_j}^{lo}\} \tag{16}$$

Example 5. *As shown in the Figure 6, the concurrent nodes of v_1 is $C(v_1) = \{v_3, v_4\}$. As the priority of v_1 ($P_{v_1} = 3$) is higher than the priorities of v_3 ($P_{v_3} = 1$) and v_4 ($P_{v_4} = 2$), we have $I_{v_1}^{lo} = \{v_3, v_4\}$. For a system with two cores, the maximum number of low-priority tasks that node v_1 can execute concurrently with is 1 and given that $W(v_3) < W(v_4)$, the intra-task interference of low-priority nodes is $I_{v_1}^{lomax} = \{v_4\}$ in this example.*

According to Example 5, $C(v_1) = \{v_3, v_4\}$ and $I_{v_1}^{lomax} = \{v_4\}$, $C(v_2) = \{v_4\}$ and $I_{v_2}^{lomax} = \{v_4\}$. In the present instance, v_4 will only be counted as interference of v_1 and not for v_2, since v_1 is the predecessor of v_2 and Equation 9 excludes repeated interference from being counted more than once. However, there may exist a situation that v_3 delays v_1 and v_4 delays v_2, as this situation still meets the constraint that each node within a DAG can suffer delay from at most $m-1$ low-priority node mentioned above. We can assume v_3 will interfere with v_1 as well to derive a safe upper bound for the intra-task interference. Such nodes are denoted as $I_{v_j}^{lopre}$ and are bounded in Equation 17. The $I_{v_j}^{lopre}$ iterates through nodes in $I_{v_j}^{lomax}$ denoted as v_p, and identifies each node that belongs to $I_{v_j}^{lo}$ but is not in $I_{v_j}^{lomax}$ (i.e., $v_k \in (I_{v_j}^{lo} \setminus I_{v_j}^{lomax})$), and is an ancestor of v_p (i.e., $v_k \in anc(v_p)$).

$$I_{v_j}^{lopre} = \bigcup_{v_p \in I_{v_j}^{lomax}} \{v_k | v_k \in (I_{v_j}^{lo} \setminus I_{v_j}^{lomax}) \wedge v_k \in anc(v_p) \wedge v_k \in V\} \tag{17}$$

Sustainability: We claim that the above analysis of the worst-case makespan for a single DAG task is sustainable [21], i.e., when a node executes less than its WCET, the makespan will not exceed the computed worst-case bound. In the constructed analysis, the worst-case makespan is obtained by finding the worst-case finish time of each node in a DAG. According to the Equations 8 and 9, the worst-case finish time of each node consists of three factors: 1) the finish time of the predecessor, 2) the associated intra-task interference, and 3) the WCET of the node.

Lemma 7. *The constructed analysis is sustainable if the intra-task interference is not increased when the predecessors of a node v_j execute less than their WCETs.*

Proof. We focus on one factor at a time to prove this lemma. First, if v_j itself executes less than its WCET, it can only result in a smaller F_{v_j}. Second, without considering the intra-task interference, given that the predecessors of v_j execute less than their WCETs, then $F(v_j)$ can only demonstrate a monotonically non-increasing trend. Therefore, any sustainability issue in the constructed analysis could only occur when the predecessors of v_j execute less than their WCET but have caused an increased intra-task interference. □

Theorem 2. *The constructed worst-case response time analysis is sustainable.*

Proof. Following Lemma 7, we focus on proving that for a node v_j, its intra-task interference will not increase when its predecessor nodes execute less than their WCETs. First, the set of concurrent nodes of v_j will remain the same regardless of the actual execution time v_j takes, which is derived purely based on execution dependency. Then, the list $I_{v_j}^{remove}$ cannot be affected as the nodes are accounted for based on the structural characteristics of the DAG instead of the execution time of the nodes (Theorem 1). This precludes $I_{v_j}^{potential}$ from being affected (Equation 12). To this end, we proved that the list $I_{v_j}^{potential}$ cannot be affected by a lower execution time of nodes in a DAG. Based on Equation 13, the intra-task interference is determined by $I_{v_j}^{hi}$, $I_{v_j}^{lomax}$, and $I_{v_j}^{lowpre}$. However, as shown in Equations 14, 16, and 17, these factors are computed based on the priorities and the WCETs of the nodes, which provide the maximum possible bound. Therefore, any node with a lower execution time will not increase the intra-task interference of v_j, and hence, the theorem follows. □

[2]In a multi-DAG scenario, the source node may experience interference from up to m low-priority nodes, while the remaining nodes may experience interference from up to $m-1$ low-priority nodes [10], [20].

B. Response time for multi-DAG systems

In this section, we extend the analysis from a single DAG to a multi-DAG system, in which there exists a set of recurrent DAG tasks $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. The worst-case response time of τ_i is denoted as R_i. To calculate R_i, the goal is to determine the worst-case finish time of each node in τ_i (i.e. $F(v_{i,j}), \forall v_{i,j} \in V_i$) which follows the same principle as Section V-A. The only difference is that each node will incur interference not only from nodes within the same DAG but also from other DAGs, i.e., inter-task interference. To compute this interference, the identification of the concurrent nodes (see Equation 1) is updated to include nodes from other DAG tasks, denoted as $C^*(v_{i,j})$ for the multi-DAG case in Equation 18.

$$C^*(v_{i,j}) = C(v_{i,j}) \cup \bigcup_{\tau_k \in \Gamma} \left\{ \left\lceil \frac{R_i}{T_k} \right\rceil * V_k \middle| v_{i,j} \notin V_k \right\} \quad (18)$$

As shown in this equation, the set of concurrent nodes of $v_{i,j}$ in a multi-DAG system contains 1) the concurrent nodes from the same DAG (i.e. $C(v_{i,j})$ in Equation 1 and the nodes from other DAGs, which is denoted as $\bigcup_{\tau_k \in \Gamma} \{\lceil \frac{R_i}{T_k} \rceil * V_k | v_{i,j} \notin V_k\}$. Notation $\lceil \frac{R_i}{T_k} \rceil$ provides the number of releases of τ_k during R_i and V_k represents all nodes in τ_k. The multi-DAG analysis follows the same process as the single-DAG analysis described in Section V-A but includes extra inter-task interference imposed by other DAG tasks. The key principle is still to work out the worst-case execution scenario of each node within a DAG.

Instead of assuming that each node incurs a fixed amount of interference as mentioned in Section IV. The proposed analysis takes into account the actual worst-case execution scenario to count inter-task interference. The resulting analysis leads to a reduction in unnecessary interference and facilitates the full exploration of the parallelism between nodes. This allows our analysis to alleviate Limitation I and II summarized in Section IV.

VI. Evaluation

In this section, we compare the tightness of the analytical bounds between the proposed and the state-of-the-art methods. The evaluation focuses on comparing the worst-case completion of a single DAG (i.e., the makespan) and the overall system schedulability of multi-DAG systems.

Experimental Setup. Each DAG task in the evaluation is generated using two structural parameters: the *length* indicating the number of node layers required during generation, and the *parallelism* defining the number of nodes to be generated in a layer. Starting from a single source node, nodes in a DAG task are generated layer by layer given the *length* and the *parallelism* settings, where the node generation is finished by adding a single sink node. Each newly-generated node has a probability of 50% to be connected to existing ones. Finally, nodes without any predecessor (or successor) are directly connected to the source (or the sink) node.

For a single recurrent DAG task τ_i, its period T_i is randomly determined within the range of 1000 to 2000 milliseconds, and its deadline is set equal to the period. The utilization of τ_i is set to $U_i = 0.5$. As for the experiments for multi-DAG systems, each system contains eight DAGs that are generated as described above. The total utilization of the system is determined by 8 times a basic utilization value for each task, e.g. $8 \times 0.5 = 4$. The actual utilization of each DAG τ_i is computed using the UUnifast Algorithm [22]. When each DAG task is generated with a utilization, the total WCET of each DAG task τ_i is calculated by $W(\tau_i) = U_i \times T_i$. The WCET of each node (i.e., $W(v_i)$ for a node v_i) is then randomly distributed by $W(\tau_i)$, where we enforce $W(v_i) \geq 0$. In addition, the intra-task priority assignment in [12] is applied for nodes in the DAG. The deadline Monotonic Priority Ordering is used to assign the DAG-level priorities.

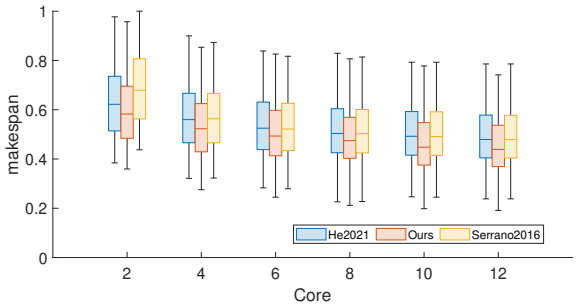

Fig. 7: The makespan of a single DAG with a varied number of cores, Parallelism = 8 and Length = 7.

For each system setting in one experiment, 1000 trials are generated for evaluation and the makespan results for single DAGs are normalized against the maximum value observed in the experiment (where applied). Each DAG is analyzed using the following competing methods.

- The state-of-the-art generic bound for the limited preemption scheme in [10] (denoted as *Serrano2016*).
- The state-of-the-art priority-explicit bound proposed in [12] with the blocking terms in [10] applied (i.e., $I_{v_j}^{lo}$ and $I_{v_j}^{lopre}$ in Equation 16 and 17) for use under the limited preemption scheme[3] (denoted as *He2021*).
- The response time analysis constructed in this paper (denoted as *ours*).

A. Evaluation of the makespan for a single DAG

Figures 7 to 9 present the worst-case makespan of a single DAG with a varied number of cores, the degree of parallelism and length, respectively. From Figure 7, we can observe that the proposed analysis demonstrates a constant lower bound on the makespan compared to other methods. For instance, the proposed analysis outperforms He2021 and Serrano2016 by an average of 8.8% and 8.47% under 6 cores, respectively. The reason for this observation is that the proposed analysis explicitly considers the parallel execution of nodes in a DAG

[3]We note that this modification is necessary for the application on a limited preemption scheduler and the modified analysis is still the state-of-the-art analysis with explicit priorities under the target scheduling scheme.

TABLE I: The Comparison of the worst-case makespan calculated by the proposed method and He2021.

	Our $\succ$ He2021						Our $\prec$ He2021					
	m=2	m=4	m=6	m=8	m=10	m=12	m=2	m=4	m=6	m=8	m=10	m=12
number	N= 981	N=995	N=991	N=992	N=1000	N= 1000	N=5	N=5	N=9	N=8	N=0	N=0
avg.	7.21	9.17	8.88	9.24	8.04	6.75	0.06	1.65	1.52	1.86	0	0
max.	21.64	27.68	26.47	22.14	23.91	18.22	0.3	3.4	2.44	3.84	0	0

TABLE II: The Comparison of the worst-case makespan calculated by the proposed method and Serrano2016.

	Our $\succ$ Serrano2016						Our $\prec$ Serrano2016					
	m=2	m=4	m=6	m=8	m=10	m=12	m=2	m=4	m=6	m=8	m=10	m=12
number	N= 1000	N=987	N=980	N=991	N=1000	N= 1000	N=0	N=13	N=20	N=9	N=0	N=0
avg.	13.68	9.11	8.47	8.94	7.91	6.69	0	1.3	1.92	1.77	0	0
max.	28.29	26.75	22.01	20.56	19.28	16.87	0	3.06	5.21	3.84	0	0

and only takes ones that can cause a delay, and hence, leads to lower intra-task interference.

Similar observations are also obtained in Figures 8 and 9, in which the proposed analysis is constantly better than the competing methods in terms of achieving tighter bounds on the makespan. From these figures, a notable observation is that the proposed analysis performs better when the length of the DAG is relatively low, e.g., with a length less than seven in Figure 9. This is because with a shorter length, fewer layers of nodes are generated so that less nodes will be included in $I_{v_j}^{potential}$. According to Equation 13, if nodes in $I_{v_j}^{potential}$ cannot take up all the cores, v_j can start execution without incurring any interference. However, in the state-of-the-art methods, such workload is still accounted for when bounding the intra-task interference, and hence, results in a longer makespan.

In addition, to provide a more detailed comparison of the evaluated methods, Tables I and II show the number of cases in which our method demonstrates an improvement over the other methods, and the percentage of improvement in such cases, for *our* versus *He2021* and *our* versus *Serrano2016*, respectively. From the tables, we can observe that the proposed analysis outperforms other schemes across all settings, and obtains an improvement of up to 9.24% and 13.68% on average, compared to He2021 and Serrano2016 respectively. However, we also observed that there exist cases where He2021 or Serrano2016 can produce a lower makespan bound. This is because in order to provide a safe bound, the $I_{v_j}^{lopre}$ in Equation 13 is introduced in our analysis to avoid over-optimistic computations when bounding the delay from low-priority nodes. However, such situations rarely occur and the proposed analysis demonstrates a significant advantage over the competing analysis in almost all cases.

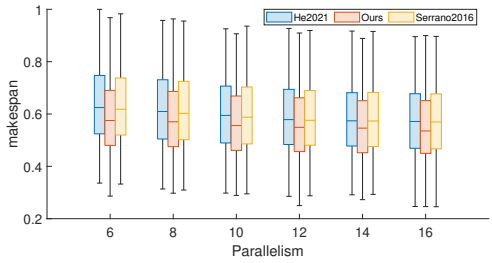

Fig. 8: The makespan of a single DAG with varied degree of parallelism, Core = 6 and Length = 7.

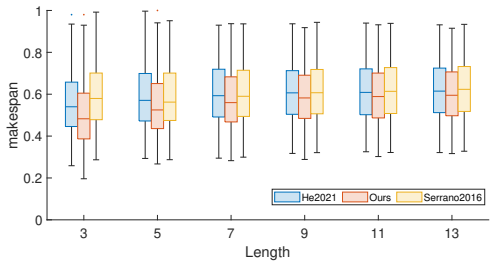

Fig. 9: The makespan of a single DAG with varied length, Core = 6, and Parallelism = 12.

B. Evaluation of system schedulability for multi-DAG systems

Figures 10 and 11 present the schedulability of the evaluated methods in multi-DAG systems, with a varied number of cores and utilization per core. In the multi-DAG case, the proposed analysis demonstrates the most pronounced advantages, in which the schedulability of other methods drops quickly and schedules few systems when either the number of cores is lower than 14 or the utilization is higher than 0.2. In contrast, the proposed analysis can still schedule up to 100% of the systems in majority cases (e.g., under 12 cores in Figure 10 and with a utilization per core of 0.5 in Figure 11). This observation is obtained due to a combinational effort on reducing the bound of both the intra-task and inter-task interference in the proposed analysis. Especially, when computing the inter-task interference, the proposed analysis reduces the pessimism when accounting for the number of nodes that contribute to inter-task interference by fully exploring the execution parallelism between nodes. However, as for He2021 and Serreno2016, the total workload of the interfering DAGs is used when computing the inter-task interference, along with repetitive computations when bounding the low-priority interference (i.e., Limitation II discussed in Section IV). More specifically, in the single-DAG analysis, we focus on calculating the worst-case scenario of nodes instead of paths which

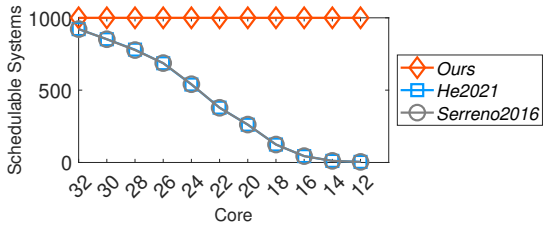

Fig. 10: The system schedulability for multi-DAGs under a varied number of cores, Parallelism = 6, and Length = 6.

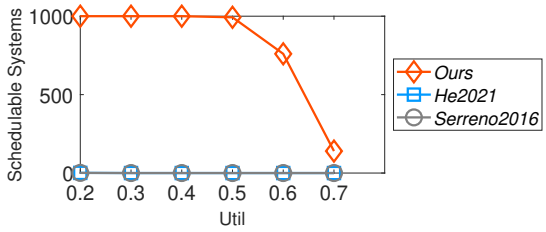

Fig. 11: The system schedulability for multi-DAGs under varied utilization per core, Core = 4, Parallelism = 6, and Length = 6.

allow a finer-grained interference analysis. The key factor for better schedulability is the removal of unnecessary interference delay that we summarized in Equation 11 and the condition that judges whether interference will occur in Equation 13. Moreover, in the multi-DAG analysis, we continue to analyze the worst-case execution scenario of each node and treat other DAGs as concurrent nodes as shown in Equation 18 which precisely captures the inter-task interference.

We also observed that *He2021* and *Serreno2016* have almost the same schedulibility in Figures 10 and 11. The reason is that the work in [12] provides a bound for single-DAG preemptive scheduling which can have an obvious advantage over generic bound because only high-priority nodes are accounted as interference. However, we introduce non-preemptive (our focus) features to *He2021* which includes low-priority interference, and the advantage becomes trivial. Moreover, *He2021* inherits the analysis of Serreno2016 in multi-DAG, hence their overall difference is negligible with a non-preemptive scheduling scheme.

VII. Concluding Remarks

This paper studies the response time analysis of DAG tasks under a global limited preemption scheme. An in-depth analysis is first proposed to show the limitations and defects of the existing analyzing methods. Inspired by the discussion, a new response time analysis for a single DAG task and multi-DAG systems is constructed, which precisely accounts for the intra-task and inter-task interference by fully exploring the parallelism between nodes. Experimental results have demonstrated the tightness of the constructed analysis over the state-of-the-art methods. In future work, we aim to extend the constructed analysis to support other scheduling methods (e.g., fully-partitioned and federated) and heterogeneous architectures.

Acknowledgment

This research was funded in part by Innovate UK HICLASS project (113213). EPSRC Research Data Management: No new primary data was created during this study.

References

[1] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate DAGs from multi-rate task sets," in *Real-Time and Embedded Technology and Applications Symposium*, 2020.

[2] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *Real-Time Systems Symposium*, 2012.

[3] A. Burns and A. J. Wellings, *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*, 2001.

[4] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional dag tasks in multiprocessor systems," in *IEEE Euromicro Conference on Real-Time Systems*, 2015.

[5] J. C. Fonseca, V. Nélis, G. Raravi, and L. M. Pinho, "A multi-DAG model for real-time parallel applications with conditional execution," in *Annual ACM Symposium on Applied Computing*, 2015.

[6] J. Fonseca, G. Nelissen, V. Nelis, and L. M. Pinho, "Response time analysis of sporadic DAG tasks under partitioned scheduling," in *Symposium on Industrial Embedded Systems*, 2016.

[7] S. Baruah, "The federated scheduling of systems of conditional sporadic DAG tasks," in *International Conference on Embedded Software*, 2015.

[8] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, 2011.

[9] J. Fonseca, G. Nelissen, and V. Nélis, "Improved response time analysis of sporadic DAG tasks for global FP scheduling," in *International Conference on Real-Time Networks and Systems*, 2017.

[10] M. A. Serrano, A. Melani, M. Bertogna, and E. Quiñones, "Response-time analysis of DAG tasks under fixed priority scheduling with limited preemptions," in *Design, Automation & Test in Europe Conference & Exhibition*, 2016.

[11] Q. He, N. Guan, Z. Guo *et al.*, "Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[12] Q. He, M. Lv, and N. Guan, "Response time bounds for dag tasks with arbitrary intra-task priority assignment," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, 2021.

[13] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "Dag scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *IEEE Real-Time Systems Symposium*, 2020.

[14] S. Zhao, X. Dai, and I. Bate, "Dag scheduling and analysis on multi-core systems by modelling parallelism and dependency," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[15] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *Real-Time Systems Symposium*, 2017.

[16] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *IEEE Real-Time Systems Symposium*, 2018.

[17] M. Hatami, "Semi-partitioned scheduling hard real-time periodic dags in multicores," in *The Proceeding of First Work-in-Progress Session of 2018 CSI International Symposium on Real-Time and Embedded Systems and Technologies*, 2018.

[18] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *Euromicro Conference on Real-Time Systems*, 2014.

[19] R. L. Graham, "Bounds on multiprocessing timing anomalies," *Journal on Applied Mathematics*, 1969.

[20] A. Thekkilakattil, R. I. Davis, R. Dobrin, S. Punnekkat, and M. Bertogna, "Multiprocessor fixed priority scheduling with limited preemptions," in *International Conference on Real Time and Networks Systems*, 2015.

[21] A. Burns and S. K. Baruah, "Sustainability in real-time scheduling." *Journal of Computing Science and Engineering*, 2008.

[22] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, 2005.