Parallel Path Progression DAG Scheduling

Niklas Ueter, *Student Member, IEEE*, Mario Günzel, *Student Member, IEEE*, Georg von der Brüggen, *Senior Member, IEEE*, and Jian-Jia Chen, *Senior Member, IEEE*

***Abstract*—Increasing performance needs of modern cyber-physical systems leads to multiprocessor architectures being increasingly utilized. To efficiently exploit their potential parallelism in hard real-time systems, appropriate task models and scheduling algorithms that allow to provide timing guarantees are required. Such scheduling algorithms and the corresponding worst-case response time analyses usually suffer from resource over-provisioning due to pessimistic analyses based on worst-case assumptions. Hence, scheduling algorithms and analyses with high resource efficiency are required. A prominent fine-grained parallel task model is the directed-acyclic-graph (DAG) task model that is composed of precedence constrained subjobs. This paper studies the hierarchical real-time scheduling problem of sporadic arbitrary-deadline DAG tasks. We propose a parallel path progression scheduling property that is implemented with only two distinct subtask priorities, which allows to quantify the parallel execution of a user chosen collection of complete paths in the response time analysis. This novel approach significantly improves the state-of-the-art response time analyses for parallel DAG tasks for highly parallel DAG structures and can provably exhaust large core numbers. Two hierarchical scheduling algorithms are designed based on this property, extending the parallel path progression properties and improve the response time analysis for sporadic arbitrary-deadline DAG task sets.**

***Index Terms*—Real-time DAG scheduling, homogeneous multicore platforms, hierarchical scheduling, approximation algorithms.**

I. Introduction & Motivation

MODERN cyber-physical systems have shifted from uniprocessor to multiprocessor systems in order to deal with thermal and energy constraints, as well as the computational demands of increasingly complex applications with tight deadline constraints. Notably, the number of available cores in multicore architectures that are meant to be used in the real-time domain has greatly increased. This is exemplified by the *Kalray MPPA* architecture with 256 cores, exacerbating the need for methodologies that are capable to utilize the potential of the available cores. This architectural shift poses multiple challenges for the design methodology of real-time aware parallel software, the specification and implementation of fine-grained parallel task models, and the design of appropriate scheduling algorithms that allow for formal response time analyses.

The de-facto standard programming and scheduling model for parallel computing OpenMP and the real-time extension OmpSs [14] are using hierarchical scheduling. That is, on the higher-level, the worker threads are scheduled by the operating system and on a lower level the subjobs of the parallel tasks are managed by the respective runtime environment that is responsible to dispatch ready subjobs to the available worker threads. The concrete implementation of the hierarchical scheduling are not standardized, e.g., the approaches of OpenMP and OmpSs differ in that, OpenMP implements fork-join parallelism with a master thread that creates a so-called team of parallel threads on encountering a parallel region. In contrast, OmpSs uses a pool of worker threads to serve the subjobs as soon as they become ready, which is similar to list scheduling. In both frameworks, the application source code is written in a high-level programming language, which is instrumented with a set of directives, that, together with library routines and a provided runtime environment are used to describe and execute the parallel applications.

We believe that the decoupling of parallel application design and scheduling on the one hand, and real-time operating system scheduling and service contracts on the other hand, is the most promising approach to implement real-time parallel software due to temporal isolation and simple integration with any commonly available scheduling algorithm of the used real-time operating system such as partitioned or global scheduling variants of fixed-priority or earliest-deadline first (EDF) scheduling algorithms.

The response-time analysis of the hierarchical scheduling approach is decomposed into the problem to verify that each of the worker threads is able to provide a defined amount of service – that must be promised by the real-time operating system – and the subsequent problem to verify that each DAG job can finish its workload with the promised service within its deadline constraints.

In the real-time scheduling theory of the DAG task sets, the objective is to efficiently utilize the parallelism provided by multiprocessors for task sets with inter- and intra-task parallelism, while guaranteeing that each task meets its deadline. Parallelism can be categorized into inter-task parallelism, which refers to the parallel execution of distinct tasks, each of which executes sequentially and intra-task parallelism which refers to the parallel execution of a single task. Intra-task parallelism requires task models with subtask level granularity that can be scheduled in parallel, e.g., Fork-join models [25], synchronous parallel task

Manuscript received 4 October 2022; revised 9 April 2023; accepted 16 May 2023. Date of publication 25 May 2023; date of current version 6 September 2023. This work was part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under Grant 865170. This work was supported by Deutsche Forschungsgemeinschaft (DFG), as part of Sus-Aware under Grant 398602212. Recommended for acceptance by E. Bini. *(Corresponding author: Jian-Jia chen.)*

The authors are with the Design Automation for Embedded Systems Group, TU Dortmund University, 44227 Dortmund, Germany (e-mail: niklas.ueter@tu-dortmund.de; mario.guenzel@tu-dortmund.de; Georg.von-der-brueggen@tu-dortmund.de; jian-jia.chen@tu-dortmund.de).

Digital Object Identifier 10.1109/TC.2023.3280137

models, or DAG (directed-acyclic graph) based task models. A plethora of real-time scheduling algorithms and response time analyses thereof have been proposed, e.g., for generalized parallel task models [33], and for DAG (directed-acyclic graph) based task models [3], [5], [13], [16], [17], [19], [28], [42]. For DAG-based task models, improvements in the response time analyses can be categorized into analyses that improve inter-task interference, e.g., in [13], [16], or intra-task interference as e.g., in [19], [20], [26], [42]. In general, intra-task interference analyses build upon the interference analysis along the execution of the envelope (also known as critical path or key path). Intuitively, the envelope path is a schedule dependent sequence of subjobs, with the property that the arrival and finishing time intervals of all envelope path subjobs are a partition of the arrival time and finishing time interval of the DAG job. Since these subjobs *envelope* the execution of the DAG job, the cumulative amount of time used to execute envelope path subjobs and the cumulative amount of time that an envelope path subjob is interfered with, bounds the response time.

In contrast to the state-of-the-art, we analyze the simultaneous progression a number of user-chosen paths of at most the number of processors alongside the envelope path. Therefore, we only have to account for the interference of subjobs that do not belong to any of those user-chosen paths for a response-time bound and are thus able to generalize Graham's makespan bound from one path to multiple paths. The improvement is achieved by intra-task prioritization, namely, by assigning a lower-priority to all subtasks of the user-chosen paths. Thereby, the progression of the envelope path subjobs and the progression of subjobs from the user-chosen paths can be analytically related to one another. To the best of our knowledge, this is the first paper that proposes the parallel path progression concepts[1] and corresponding response time analyses as well as extensions of these concepts to hierarchical scheduling.

Contributions: We provide the following contributions:

- Based on the proposed *Parallel Path Progression Concepts* in Section III-A, we propose a preemptive fixed-priority scheduling algorithm and a sustainable response time analysis for an arbitrary collection of paths of at most the number of processors in Section III-B. In Section IV, we provide a polynomial time algorithm that either finds a path collection that fully covers the DAG if one exists for the available number of processors or an approximation with a provably bounded worst-case response time.
- We extend our findings to two hierarchical scheduling algorithms in Section V. Namely a sporadic arbitrary-deadline gang reservation system in Section V-A and a sporadic arbitrary-deadline ordinary reservation system in Section V-B that make use of the *Parallel Path Progression Concepts*. The hierarchical scheduling algorithm can be applied to sporadic arbitrary-deadline DAG tasks, which may be executed concurrently with tasks described by a different task model, e.g., sequential tasks.
- For both reservation systems, we provide response time analyses and algorithms to generate "feasible" reservation systems as explained in Sections V-A and V-B, respectively.
- In Section VI, we evaluate our approach using synthetically generated DAG task sets and demonstrate that our approach advances the state-of-the-art in high-parallelism scenarios and show that the performance of our approach is between the start-of-the-art and federated scheduling in more sequential scenarios.

[1]While under submission the paper *Bounding the Response Time of DAG Tasks Using Long Paths* was published in RTSS 2023 with a similar concept. However both papers are different in the analysis approach and that we consider preemptive scheduling and hierarchical scheduling that also works for arbitrary-deadlines.

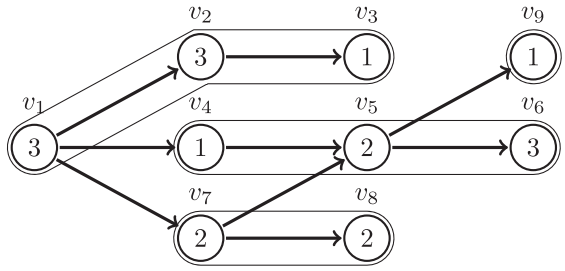


Fig. 1. An exemplary directed-acyclic graph (DAG) with subtasks $v_1, v_2, \ldots, v_9$. The numbers within the vertexes denote the subjob's worst-case execution time. The arrows represent the precedence constraints indicating that the release of a subjob depends on the finishing of all incident subjobs. The boxed vertexes denote a minimal vertex-disjoint path decomposition of G.

II. Task Model and Problem Description

We consider a set $\mathbb{T} := \{\tau_1, \ldots, \tau_n\}$ of sporadic arbitrary-deadline directed-acyclic graph (DAG) tasks that are scheduled and executed upon M homogeneous processors. Each task $\tau_i := (G_i, D_i, T_i) \in \mathbb{T}$ is defined by a DAG G_i describing the subtasks and precedence constraints, minimal inter-arrival time T_i, and relative deadline D_i. Each task releases an infinite sequence of task instances, called jobs. We use J_i^ℓ to denote the ℓ-th job of task τ_i, and a_i^ℓ, f_i^ℓ, and $d_i^\ell = a_i^\ell + D_i$ to refer to the arrival time, finishing time, and (absolute) deadline of job J_i^ℓ.

DAG: The task's DAG G_i is defined by the tuple (V_i, E_i), where V_i denotes the finite set of subtasks and the relation $E_i \subseteq V_i \times V_i$ denotes the precedence constraints among them, such that there are no cyclic precedence constraints. To be mathematically precise, each job J_i^ℓ is associated with an instance of the DAG G_i^ℓ with corresponding ℓ-th subjobs v_j^ℓ where v_j is a subtask in V_i. A subjob of the ℓ-th job of task τ_i, namely v_j^ℓ for $v_j \in V_i$, is released when all ℓ-th subjobs v_k^ℓ for $(v_k, v_j) \in V_i$ have finished execution. To reduce this notation, we drop the index of the task as well as of the job when analyzing one specific job. That is, we refer to $G = (V, E)$ and $v_j \in V$ to denote a subjob of a specific DAG job. An exemplary DAG is illustrated in Fig. 1.

Volume: The volume $vol_i : V_i \to \mathbb{R}_{\geq 0}$ specifies the worst-case execution time of each subtask $v_j \in V_i$, which means that no subjob (instance) v_j^ℓ ever executes for more than $vol_i(v_j)$ time-units on the execution platform, but may finish earlier. Moreover, the volume of any subset of subtasks $W \subseteq V_i$ is

TABLE I
SUMMARY OF USED NOTATION

SYMBOL	MEANING
$vol(v) \in \mathbb{R}$	Volume (worst-case execution time) of subtask v
$\Pi(v) \in \mathbb{N}$	Fixed-priority value of subtask v
$\Psi(G)$	All paths from source vertex to sink vertex in G
$\pi \in \Psi(G)$	Path from source to sink vertex in G
$\pi_e \in \Psi(G)$	Envelope path of G in a schedule
$\pi_* \in \Psi(G)$	Longest path in G
$\mathcal{P}(\Psi(G))$	Path collections of G, i.e., power-set of all paths
$\psi \in \mathcal{P}(\Psi(G))$	A path collection
$C \in \mathbb{R}$	Total volume $vol(V)$ of a DAG G
$V_s(\psi)$	Set of vertexes (subtasks) in the path collection ψ
$vol(\pi_*) \in \mathbb{R}$	Volume of longest path π_*
$V_s^c(\psi)$	Path-collection complement $\{v \in V \mid v \notin V_s(\psi)\}$
$\mathcal{G}_i$	Gang reservation system for $\tau_i \in \mathbb{T}$
$\mathcal{O}_i$	Ordinary reservation system for $\tau_i \in \mathbb{T}$
$w \in \mathbb{N}$	Upper-bound for the minimal size of a path cover
$\mathbb{T}$	DAG task set

$vol(W) := \sum_{v_j \in W} vol_i(v_j)$. In particular, the *total volume* of a task τ_i is given by $C_i := vol_i(V_i)$.

Release & Deadline: In real-time systems, tasks must fulfill timing requirements, i.e., each job J_i^ℓ must finish its total volume between the arrival of a job at a_i^ℓ and that job's absolute deadline at $a_i^\ell + D_i$. A task τ_i is said to meet its deadline if each job meets its deadline, i.e., $f_i^\ell \leq a_i^\ell + D_i$ for all $\ell \in \mathbb{N}$. We consider *arbitrary-deadlines*, which means that we do not make any assumptions about the relation of deadline and inter-arrival time. For example, the relative deadline may be less than the minimal inter-arrival time ($D_i \leq T_i$) in which case a new job is only released if the previous job is finished. Alternatively, the deadline can be larger than the minimal inter-arrival time ($D_i > T_i$) in which case a new job can be released despite an unfinished prior job. The release times of any two subsequent jobs of τ_i is at least T_i time-units apart, i.e., $a_i^{\ell+1} \geq a_i^\ell + T_i$ for all $\ell \in \mathbb{N}$. A summary of used notation is provided in Table 1.

III. PARALLELISM & PATH-AWARE SCHEDULING

The parallelism of a DAG task is inherently limited by the paths that it is composed of, since a path enforces a sequential execution order of the associated subjobs. In this paper, we aim to reduce intra-task interference by enforcing properties that track and guarantee parallel progress of a collection of paths within a DAG and thus allow to significantly improve the worst-case response time for high-parallel use cases. To that end, we answer the following questions: (*Q1*) What are the minimal theoretical properties required to track and guarantee parallel path progression on a set of dedicated processors? (*Q2*) Can any collection of paths be used for parallel progression and if so what is a provably good selection? (*Q3*) How can the results from (*Q1*) and (*Q2*) be extended to consider hierarchical scheduling?

A. Parallel Path Progression Concepts

In this subsection, we examine the required properties to achieve parallel path progression on M processors dedicated to execute a single job of a DAG task. By that, we avoid any inter-task interference and solely focus on intra-task interference.

Definition 1 (Path): For each subtask $v_j \in V$ of a DAG $G = (V, E)$ the set of predecessors of v_j is given by $pred(v_j) := \{v_i \in V \mid (v_i, v_j) \in E\}$. Respectively, the set of successors of v_j is given by and $succ(v_j) := \{v_i \in V \mid (v_j, v_i) \in E\}$. A path is an ordered set of subtasks $\pi := \langle v_1, \ldots, v_n \rangle$ such that $pred(v_1) = \emptyset$, $succ(v_n) = \emptyset$ and $v_k \in pred(v_{k+1})$ for all $k \in \{1, \ldots, n-1\}$. If either $pred(v_1) \neq \emptyset$ or $succ(v_n) \neq \emptyset$ then π is not considered a path in the context of this paper.

Definition 2 (n-Path Collection): Let a DAG $G = (V, E)$ then the enumeration of all possible paths is denoted as $\Psi(G) := \{\pi \mid \pi \text{ is a path according to Definition 1 in} G\}$. Any subset of paths $\psi \in \mathcal{P}(\Psi(G))$ from the powerset of $\Psi(G)$ is called a *path collection*. Further, a path collection $\psi \in \mathcal{P}(\Psi(G))$ is called an *n-path collection* if $|\psi| = n$, i.e., ψ is a collection of n-many paths.

In the remainder of this paper we will use π_* to denote the longest path in G, i.e., $vol(\pi_*) \geq vol(\pi)$ for all $\pi \in \Psi(G)$. It is a fact that the maximal number of paths that can be executed in parallel is limited by the number of processors M. Therefore we constrain our solution space to n-path collections where $n \in \{1, \ldots, M\}$. Based on a concrete n-path collection ψ, the set of subtasks that belong to at least one of the paths in ψ is defined by $V_s(\psi) := \pi_{\psi_1} \cup \cdots \cup \pi_{\psi_n}$ for each $\pi_{\psi_1}, \ldots, \pi_{\psi_n} \in \psi$. Please note that we use the subscripts to index the paths belonging to the path collection. Conversely, the complement set of subtasks that do not belong to any of the selected paths is denoted by $V_s^c(\psi) := \{v \in V \mid v \notin V_s(\psi)\}$.

We propose a parallel-progress prioritization that gives each subtask a priority based on the membership of the above sets, which is formalized in the following definition. Later in this section, we explain how this prioritization can be used to better analyze the self-interference by explicitly considering the parallel execution of paths in ψ in the response-time analysis.

Definition 3 (Parallel Path Progression Prioritization): Let $V_s(\psi)$ denote the set of subtasks from an n-path collection ψ of a DAG $G = (V, E)$. A fixed-priority policy for all subtasks $v \in V$ is a *parallel path progression prioritization* if and only if $\Pi(v_i) < \Pi(v_k)$ for any two $v_i \in V_s(\psi)$ and $v_k \in V_s^c(\psi)$, where $\Pi(v_i)$ denotes the priority of subtask v_i.

Note that in our notation for the priorities, a higher value implies a higher priority, i.e., $\Pi(v_i) > \Pi(v_k)$ implies that v_i has a higher priority than v_k. A sufficient policy to satisfy the parallel path progression prioritization property is to only use two distinct priority-levels.

We clarify the introduced notation and definitions collectively in the following example. The path enumeration $\Psi(G)$ of the DAG illustrated in Fig. 1 consists of six paths $\{\pi_1, \pi_2, \ldots, \pi_6\}$, namely; $\pi_1 := \langle v_1, v_2, v_3 \rangle$, $\pi_2 := \langle v_1, v_4, v_5, v_9 \rangle$, $\pi_3 := \langle v_1, v_4, v_5, v_6 \rangle$, $\pi_4 := \langle v_1, v_7, v_5, v_9 \rangle$, $\pi_5 := \langle v_1, v_7, v_5, v_6 \rangle$, and $\pi_6 := \langle v_1, v_7, v_8 \rangle$. A 2-path collection ψ from the powerset $\mathcal{P}(\Psi(G))$ is for instance given by $\psi := \{\pi_2, \pi_3\}$. Subsequently, $V_s(\psi) = \pi_2 \cup \pi_3 = \{v_1, v_4, v_5, v_6, v_9\}$ and $V_s^c(\psi) := \{v_2, v_3, v_7, v_8\}$. If for instance all subjobs $v_i \in V_s(\psi)$ are assigned priority $\Pi(v_i) = 1$

and conversely all subjobs $v_i \in V_s^c(\psi)$ are assigned priority $\Pi(v_i) = 2$, then this prioritization is a valid parallel path progression prioritization.

B. Parallel Path Progression Scheduling

In this subsection, we look at a single DAG job that is scheduled on M dedicated processors by a work-conserving preemptive list scheduling algorithm in conjunction with the parallel path progression prioritization. We elaborate how this prioritization aids the analysis of the parallel progression of a path collection and in consequence the response time analysis of the DAG job.

Definition 4 (List-FP): In a *preemptive list-FP schedule* on M dedicated processors, a task instance (job) of a DAG task $G = (V, E)$ with a fixed-priority assignment of each subjob $v \in V$ is scheduled according to the following rules:

- A subjob arrives to the ready list if all preceding subjobs have executed until completion, i.e., the subjob arrival time a_i for each subjob v_i is given by $\max\{f_j \mid v_j \in pred(v_i)\}$. An arrived but not yet finished subjob is considered pending.
- At any time t, the M highest-priority pending subjobs are executed on the M processors and a lower-priority subjob is preempted if necessary.

First we introduce and formalize the concept of an envelope of a schedule, which is a sequence of subjobs, with the property that the arrival and finishing time intervals of all envelope path subjobs are a partition of the arrival time and finishing time interval of the DAG job.

Definition 5 (Envelope): Let S be any concrete schedule of the subjobs $V = \{v_1, \ldots, v_\ell\}$ of a given DAG job of some DAG task $G = (V, E)$. Let each subjob $v_k \in V$ have the arrival time a_k and finishing time f_k in S. We define the envelope of G in S as the collection of arrival and finishing time intervals $[a_{k_1}, f_{k_1}), [a_{k_2}, f_{k_2}), \ldots, [a_{k_p}, f_{k_p})$ for some $p \in \{1, \ldots, \ell\}$ backwards in an iterative manner as follows:

1) $k_i \neq k_j \in \{1, \ldots, \ell\}$ for all $i \neq j$.
2) v_{k_p} is the subjob in V with the maximal finishing time.
3) $v_{k_{i-1}}$ is the subjob preceding v_{k_i} with maximal finishing time, for all $i \in \{p, p-1, \ldots, 2\}$.
4) v_{k_1} is a source node, i.e., has no predecessor.

We call $\pi_e := \{v_{k_1}, v_{k_2}, \ldots, v_{k_p}\}$ the envelope path. We note that the definition of an envelope for a DAG job may not be unique if there are subjobs with the same finishing times. In that case, ties can be broken arbitrarily.

In the remainder of this subsection, we analyze the response time of a single DAG job using all the previously introduced properties. Let a fixed-priority list schedule S on M dedicated processors be generated for a single job J where all subjobs are prioritized according to the rule described in Definition 3. For the response time analysis, we analyze the time interval $[a_J, f_J)$ between the arrival time and finishing time of J by interval partitioning into busy and non-busy times. Any point in time $t \in [a_J, f_J)$ is called *busy* if an envelope subjob is executed in S at time t. Conversely, any point in time $t \in [a_J, f_J)$ is called *non-busy* if the envelope subjob is not executed. By the construction of the envelope according to Definition 5, it must be that at any point in time $t \in [a_J, f_J)$ an envelope subjob is pending, i.e., has arrived and not yet finished. In conjunction with the simple fact that t can be exclusively either *busy* or *non-busy* we know that the response time of DAG job J is given by the cumulative amount of time spent in either of these two states.

In contrast to prior work, we partition the *non-busy* times further into *parallel path* if envelope subjob $v_{k_i} \in V_s(\psi)$ and *non-parallel path* times if $v_{k_i} \in V_s^c(\psi)$ assuming an envelope path $\pi_e := \langle v_{k_1}, v_{k_2}, \ldots, v_{k_p} \rangle$ in S. The intuition of this approach is to tie the execution of an envelope subjob to the execution of subjobs of the path collection ψ, which is used and explained in the forthcoming analyses.

Theorem 6 (Preemptive Response Time Bound): The response time of a DAG job J with an arbitrary n-path collection $\psi = \{\pi_{\psi_1}, \ldots, \pi_{\psi_n}\} \in \mathcal{P}(\Psi(G))$ (of n at most M) that is scheduled on M dedicated homogeneous processors using preemptive List-FP scheduling is bounded from above by

$$R_J \leq vol(\pi_*) + \frac{vol(V_s^c(\psi))}{M - n + 1} \tag{1}$$

Proof: By the definition of the envelope (cf. Definition 5), we know that the interval $[a_J, f_J)$ of DAG job's arrival to its finishing time in a concrete preemptive list-FP schedule S can be partitioned into contiguous intervals $[a_{k_1}, f_{k_1} = a_{k_2}), \ldots, [f_{k_{n-1}} = a_{k_n}, f_{k_n})$ where $[a_{k_i}, f_{k_i})$ denotes the arrival and finishing time of subjob v_{k_i} for all $i \in \{1, \ldots, p\}$ in the envelope $\pi_e := \{v_{k_1}, v_{k_2}, \ldots, v_{k_p}\}$.

Busy Time: Considering each envelope subjob interval $[a_{k_i}, f_{k_i})$ individually for $i \in \{1, \ldots, p\}$, the amount of *busy* time is given by the execution time of v_{k_i}, which is by definition no more than $vol(v_{k_i})$. The cumulative amount of *busy* time in $[a_J, f_J)$ can be obtained by adding up the interval's individual *busy* times resulting in $vol(\pi_e)$, which is no more than the longest path $vol(\pi_*)$.

Non-Busy Time: Since our scheduling policy is work-conserving we have that whenever an envelope subjob v_{k_i} is not executing during $[a_{k_i}, f_{k_i})$ then all M processors must be busy executing non-envelope subjobs. Since v_{k_i} can be exclusively either in $V_s(\psi)$ or $V_s^c(\psi)$, we analyze the set $\{t \in [a_J, f_J) \mid \text{envelope subjob is not executing}\} \cap [a_{k_i}, f_{k_i})$ for both cases individually:

- *Parallel Path:* Let $v_{k_i} \in V_s(\psi)$ and by assumption not execute at time t then at most $n-1$ processors execute subjobs from $V_s(\psi)$. That is because all subjobs in $V_s(\psi)$ stem from n different paths, which implies that there can never be more than n-many subjobs from $V_s(\psi)$ pending concurrently in general. Moreover, since by assumption $v_{k_i} \in V_s(\psi)$ and is not executing at t at most $n-1$ subjobs from $V_s(\psi)$ are pending. Conversely, we know that at least $M - (n-1)$ processors execute subjobs from $V_s^c(\psi)$, since otherwise v_{k_i} would be executed contradicting the case assumption.
- *Non-Parallel Path:* In the other case, let $v_{k_i} \in V_s^c(\psi)$ and by assumption not be executing at time t then it must be that no processor is executing any subjob from $V_s(\psi)$. That is because if any of the lower-priority subjobs in $V_s(\psi)$ would be executing, then the higher-priority envelope subjob $v_{k_i} \in V_s^c(\psi)$ would be executing as well, which contradicts

the case assumption. Conversely, we know that all M processors are exclusively used to execute subjobs from $V_s^c(\psi)$.

In summary, we have that during all *non-busy times* $t \in [a_J, f_J)$, we have that at least $M-(n-1)$ processors execute subjobs from $V_s^c(\psi)$. The total volume of subjobs from $V_s^c(\psi)$ during $[a_J, f_J)$ is at most $vol(V_s^c(\psi))$. The maximal cumulative amount of *non-busy times* is achieved by evenly distributing the workload and is thus no more than $vol(V_s^c(\psi))/(M-(n-1))$. □

Sustainability of Our Response Time Analysis: Many multiprocessor hard real-time scheduling algorithms and schedulability analyses presented in the literature are not sustainable, which means that they suffer from timing anomalies. These anomalies describe the counter-intuitive phenomena that a job that was verified to always meet its deadline can miss its deadline by augmenting resources, e.g., to execute the job on more processors or to decrease the execution-time (early completion). In Corollary 7, we show that our response time bound is sustainable with respect to the number of processors and the subjob execution-time. This is a beneficial property in dynamic environments, where available processors and execution times vary, and ultimately simplifies implementation efforts in real systems.

Corollary 7 (Sustainability): The response time bounds in Theorem 6 hold true for a DAG job with $G=(V,E)$ even if any subjob $v \in V$ completes before its worst-case execution time or if the number of processors is increased.

Proof: This comes directly from the observation that the volume of the envelope path $vol(\pi_e)$ as well as the length of *parallel path non-busy* and *non-parallel path non-busy* intervals can only decrease if the worst-case execution time of any subjob decreases or the number of processors is increased. Since the response time is upper-bounded by the sum of times that an envelope job is executed and the sum of times that no envelope job is executed, the corollary is proved. □

IV. Path Collection Algorithm

In our approach, any n-path collection, where n is at most the number of dedicated processors M can be chosen, which then determine the worst-case response time. Ideally, we are interested in the minimal achievable makespan, i.e., the minimal worst-case response time of a single DAG job as stated in (1). In the case that $n=1$, the volume of the set $V_s^c(\psi)$ is clearly minimal if the longest path is chosen. However, finding a makespan optimal n-path collection for every given $n>1$ is to the best of our knowledge not efficiently solvable.

To that end, we propose a polynomial time approximation algorithm for the makespan optimal n-path collection problem for a given DAG, which is shown in Algorithm 1 and prove approximation bounds. Our proposed algorithm uses the theorem of Gallai and Milgram and the greedy strategy proposed for the Weighted Maximum Coverage problem [31]. The conceptual connection of these to our problem and algorithm are explained hereinafter.

A. DAG Vertex Coverage

The general problem to find the minimal number of vertex-disjoint paths – such that all vertexes of a graph G are covered – is an NP-complete problem as can be shown by reduction to the HamiltonianCycle problem. For directed graphs G however, it was shown by Gallai and Milgram in 1960 that the minimal number of vertex-disjoint paths to cover all vertexes of a directed graph G is no more than the size of the maximal independent set of G, generalizing the results from Dilworth [11] and König-Egevary [10].

In the case that the directed graph is also acyclic, i.e., a DAG, then the largest independent set of G can be computed in polynomial time by the known reduction to the maximal matching problem in bipartite graphs by using, e.g., the Hopcroft-Karp algorithm.

For instance, the set of vertex-disjoint paths that cover the DAG illustrated in Fig. 1 is given by $U=\{\langle v_1,v_2,v_3\rangle, \langle v_4,v_5,v_6\rangle, \langle v_7,v_8\rangle, \langle v_9\rangle\}$, which is calculated as described above. In our proposed path collection algorithm, we require a collection of paths from source to sink vertexes of G that fully cover G, which are not necessarily vertex-disjoint. Hence, we here explain the explicit algorithmic construction as required in line 1 of our proposed Algorithm 1 to obtain $w \in \mathbb{N}$ many paths that cover G. Please note that for our algorithm, the knowledge of the numerical value w is sufficient without knowing the explicit paths. To prove the existence however, we construct the w-many paths explicitly.

For each $i \in \{1,\dots,|U|\}$, we initialize the i-th path π_i with the vertex-disjoint path $u_i=\langle u_{i_1},\dots,u_{i_n}\rangle \in U$ and apply the following steps successively until all π_i are paths according to Definition 1:

- If the left-most vertex in π_i is not a source vertex of G, then pick any $u_h=\langle u_{h_1},\dots,u_{h_m}\rangle \in U$ such that $\langle v_{u_{h_z}}, v_{u_{i_1}}\rangle$ in E for some $z \in \{1,\dots,m\}$ and extend the path to $\pi_i = \langle u_{h_1},\dots,u_{h_z}\rangle \circ \pi_i$.
- If the right-most vertex in π_i is not a sink vertex of G then pick any tuple $u_h=\langle u_{h_1},\dots,u_{h_m}\rangle \in U$ such that $\langle v_{u_{i_n}}, v_{u_{h_z}}\rangle$ in E for some $z \in \{1,\dots,m\}$ and update the path to $\pi_i = \pi_i \circ \langle u_{h_z},\dots,u_{h_m}\rangle$

For instance, with reference to the provided example, we start at $u_4=(9)$ with $\pi_4=\langle v_9\rangle$, which is a sink vertex in G and identify tuple $u_2=(4,5,6)$ since $(v_5,v_9)\in E$ and the path is updated to $\pi_4=\langle v_4,v_5,v_9\rangle$. Since v_4 is not a source vertex in G, we continue and identify $u_1=(1,2,3)$ due to $(v_1,v_4)\in E$ and update $\pi_4=\langle v_1,v_4,v_5,v_9\rangle$. Since v_1 is a source vertex, the procedure is terminated. Repeating the procedure yields the four simple paths $\pi_1=\langle v_1,v_2,v_3\rangle, \pi_2=\langle v_1,v_7,v_8\rangle, \pi_3=\langle v_1,v_4,v_5,v_6\rangle$, and $\pi_4=\langle v_1,v_4,v_5,v_9\rangle$, which collectively cover all vertexes $v\in V$. Please note that while in this example, $w=4$ is the minimal number of paths to cover G, the algorithm in general only provides a safe upper-bound.

B. Weighted Maximum Coverage

Another related algorithm is the Weighted Maximum Coverage [31] problem. Hereinafter, we map the problem of finding an n-path collection ψ for a DAG G that maximizes $vol(V_s(\psi))$ (minimizes $vol(V_s^c(\psi))$) to that problem as follows:

Algorithm 1: n-Path Collection Approximation (nPCA).

Require: DAG $G = (V, E)$, No. CPU M, WCET vol.
Ensure: An approximately optimal n-path collection ψ
1: $\psi^* \leftarrow$ PATHCOVERAGE$(G) := \{\pi_{\psi_1}, \ldots, \pi_{\psi_w}\}$
2: **if** $w \leq M$ **then**
3: **return** (ψ^*, w);
4: create $\psi_0 \leftarrow \emptyset$;
5: $z \leftarrow \infty$;
6: $vol' \leftarrow vol$;
7: **for each** $n \in \{1, \ldots, M\}$ **do**
8: create $\psi_n \leftarrow \psi_{n-1}$;
9: $\pi_n^* \leftarrow$ use DFS(G) to search the max. vol' path;
10: $\psi_n \leftarrow \psi_n \cup \pi_n^*$;
11: **for each** $v \in \pi_n^*$ **do**
12: update $vol'(v)$ to 0;
13: $z' \leftarrow (C - vol(V_s(\psi_n))) \,/\, M - (n-1)$;
14: **if** $z' < z$ **then**
15: solution $(\psi^*, n^*) \leftarrow (\psi_n, n)$;
16: update $z \leftarrow z'$;
17: **return** solution (ψ^*, n^*);

- *Input:* A problem instance I of the WEIGHTED MAXIMUM COVERAGE problem is given by a collection of sets $S := \{S_1, \ldots, S_m\}$, a weight function ω, and a natural number k. Each set $S_i \subseteq U$ is a subset from some universe U for each $i \in \{1, \ldots, m\}$ and each element $s \in S_i$ is associated with a weight as given by the function $\omega(s)$.
- *Objective:* For a given problem instance I, the objective is to find a subset $S' \subseteq S$ such that $|S'| \leq k$ and $\sum_{s \in \{\cup \{S_i \in S'\}\}} w(s)$ is maximized.

It was shown by Nemhauser et al. [31] that any polynomial time approximation algorithm of the WEIGHTED MAXIMUM COVERAGE problem has an asymptotic approximation ratio with respect to an optimal solution that is lower-bounded by $1 - 1/e$ unless $P = NP$, where e is Euler's number. This approximation ratio can be achieved by a greedy strategy that always chooses the set which contains the largest weights of not yet chosen elements. Despite WEIGHTED MAXIMUM COVERAGE and our problem not being equivalent, we use the same approximation strategy for the n-path Collection Approximation in Algorithm 1.

C. Approximation Algorithm

On the basis of the existence of a path collection of size w, it is possible to analyze the approximation quality of Algorithm 1. We first present our proposed algorithm and thereafter prove the approximation factor.

n-Path Collection Approximation Algorithm: From line 1 to line 3 in Algorithm 1, the upper-bound w of the minimal number of paths to fully cover the is computed. If the number of processors M is sufficient to allow the parallel execution of all w paths, i.e., $w \leq M$ then those paths are chosen for the path collection.

In the other case, from line 4 to 17, in each iteration $n \in \{1, \ldots, M\}$, the longest path π_n^* with respect to the current iteration's volume function vol' is chosen. After the path is chosen, all volumes of that path's subjobs are set to 0 to indicate that the subjobs have already been covered. By this strategy, we always choose the path, which contains the largest amount of volume of not yet chosen subjobs in each iteration. Moreover, in each n-th iteration, it is probed in line 14 if the solution ψ_n strictly improves the prior solution ψ_{n-1} with one path less. At the end of the M-th iteration, an n^*-path collection ψ^* is found that yields formal guarantees as stated in Theorem 8. The maximal bipartite matching can be obtained in $\mathcal{O}(|V|)$ using the Hopcroft-Karp algorithm. The time-complexity of nPCA is dominated by the for-loop and the depth-first search (DFS) in line 9 that is invoked in each of the iterations, resulting in $\mathcal{O}(M \cdot |V||E|)$ time complexity.

Theorem 8 (nPCA): The worst-case response time of a DAG job J (makespan) on M dedicated processors using parallel path progression scheduling and for which the n^*-many paths are calculated according to Algorithm 1, is at most

$$R_{opt} \cdot \begin{cases} 1 + \frac{M}{M - n^* + 1} \cdot \left(1 - \frac{1}{w}\right)^{n^*} \leq 2 - \frac{1}{w} & w > M \geq n^* \\ 1 & M \geq w \end{cases} \tag{2}$$

where w refers to the solution of the PATHCOVERAGE.

Proof: We prove this theorem for the cases $M \geq w$ and $M < w$ individually.

Case 1: In the first case, i.e., from line 1 to line 3, let $M \geq w$. From the discussion in Section IV-A, we know that each vertex $v \in V$ of the DAG $G = (V, E)$ is covered by at least one of those w-many paths as computed by the PATHCOVERAGE algorithm, which are returned in line 3. In consequence, the response-time bound is given by

$$R_J \leq vol(\pi_*) + \frac{0}{M - (w-1)} \leq R_{opt} \tag{3}$$

which is upper-bounded by an optimal response-time, since the longest path in G is a lower-bound of any DAG job's response-time.

Case 2: Please note that w as obtained from PATHCOVERAGE is not a minimal solution, but an upper-bound of it. That is, there may exist n-many paths under the constraints $w > M \geq n \geq 1$ such that the DAGs total volume C can be covered. However such a minimal solution is not known to be computable in polynomial time. We can however prove the approximation ratio of an optimal response-time with respect to that upper-bound.

Step 1: We prove by contradiction, that for each iteration $n \in \{1, \ldots, M\}$ the following inequality holds

$$vol^{(n)}(\pi_n^*) \geq \frac{C - vol(V_s(\psi_{n-1}))}{w} \tag{4}$$

where $\psi_0 := \emptyset$ and $vol(V_s(\psi_0)) = 0$. We use $vol^{(n)}$ to refer to vol' in the n-th iteration for better clarity in this proof. Assume for contradiction that there exists an iteration $n \in \{1, \ldots, M\}$ such that

$$\forall \pi_n \in \Psi(G) \quad w \cdot vol^{(n)}(\pi_n) + vol(V_s(\psi_{n-1})) < C \tag{5}$$

holds. Then it must hold in particular that

$$w \cdot vol^{(n)}(\pi_n^*) + vol(V_s(\psi_{n-1})) < C \tag{6}$$

where - - by the algorithmic strategy – π_n^* is chosen such that for all paths $\pi \in \Psi(G)$, the inequality $vol^{(n)}(\pi_n^*) \geq vol^{(n)}(\pi)$ is satisfied. Thus, $w \cdot vol^{(n)}(\pi_n^*) \geq vol^{(n)}(\cup_{i=1}^{w} \pi_i)$ for any arbitrary collection of w-many paths. Consequently, we have that if (6) holds then

$$vol^{(n)}\left(\bigcup_{i=1}^{w} \pi_i\right) + vol(V_s(\psi_{n-1})) < C \tag{7}$$

holds as well. It is easy to see that based on the algorithm

$$vol^{(n)}\left(\bigcup_{i=1}^{w} \pi_i\right) = vol\left(\bigcup_{i=1}^{w} \pi_i\right) - vol(V_s(\psi_{n-1})) \tag{8}$$

holds, since if the volume of a vertex in the n-th iteration differs from the initial volume then that vertex must be covered by any of the collected paths during the prior iterations, i.e., $V_s(\psi_{n-1})$. Then using the identity of (8) in (7) yields that (5) leads to the condition $vol(\cup_{i=1}^{w} \pi_i) < C$ for any arbitrary collection of w-many paths, which contradictions the existence a solution of w-many paths.

Step 2: In a second step, we now claim and prove by induction that

$$vol(V_s^c(\psi_n)) = C - vol(V_s(\psi_n)) \leq \left(1 - \frac{1}{w}\right)^n \cdot C \tag{9}$$

For $n = 1$, (9) reduces to $w \cdot vol(V_s(\psi_1)) \geq C$, which holds true since we know that there exists a finite w such that $vol(\pi_{\psi_1} \cup \cdots \cup \pi_{\psi_w}) = C \leq \sum_{i=1}^{w} vol(\pi_{\psi_i}) \leq w \cdot vol(V_s(\psi_1)) = w \cdot vol(\pi_*)$, since ψ_1 only contains the longest path in G. In the induction step $n \to n+1$, we have

$$C - vol(V_s(\psi_{n+1})) = C - (vol(V_s(\psi_n)) + vol'(\pi_{n+1}^*)) \tag{10}$$

Using (4), we conclude that

$$\text{Eq. (10)} \leq C - vol(V_s(\psi_n)) - \frac{C - vol(V_s(\psi_n))}{w} \tag{11}$$

$$\leq (C - vol(V_s(\psi_n))) \cdot \left(1 - \frac{1}{w}\right) \tag{12}$$

$$\leq \left(1 - \frac{1}{w}\right)^n \cdot C \cdot \left(1 - \frac{1}{w}\right) \tag{13}$$

Conclusion: Using (9) yields that after the n-th iteration of nPCA, the maximum response time using the computed n-path collection ψ_n is at most

$$R_J \leq vol(\pi_*) + \frac{C}{M} \cdot \frac{M}{M - n + 1} \cdot \left(1 - \frac{1}{w}\right)^n \tag{14}$$

Due to the fact that $R_{opt} \geq \max\{vol(\pi_*), C/M\}$ and the minimal response time solution (ψ^*, n^*) returned by nPCA in line 17 using (14) we have that

$$R_J \leq R_{opt} \cdot \min_{n \geq 1} \left\{1 + \frac{M}{M - n + 1} \cdot \left(1 - \frac{1}{w}\right)^n\right\}$$

$$\leq R_{opt} \cdot \min_{n \geq 1} \left\{1 + n \cdot \left(1 - \frac{1}{w}\right)^n\right\} \tag{15}$$

$$\leq R_{opt} \cdot \left(2 - \frac{1}{w}\right) \tag{16}$$

Please note that (15) is due to the fact that under the constraints $(M \geq n \geq 1)$ the function $M/(M - n + 1)$ is strictly decreasing with respect to M for $n \geq 1$. Due to the constraints, the minimal feasible M is bounded from below by n and is thus no more than $n/(n - n + 1) = n$.

In addition, we have the parametric bound based on the results w and n^* generated by the algorithm namely

$$R_J \leq R_{opt} \cdot \left(1 + \frac{M}{M - n^* + 1} \cdot \left(1 - \frac{1}{w}\right)^{n^*}\right) \tag{17}$$

Finally, we have $R_J \leq$ Eq.(17) $\leq$ Eq.(16) concluding the proof.

V. Hierarchical Scheduling

We extend the properties of parallel path progression to a system with inter-task interference using a hierarchical scheduling approach. That is, the scheduling problem is separated into different scheduling levels. On the lowest level, reservation systems (or threads) are scheduled on the physical processors by some scheduling policy that is compliant with the model of the reservation system. On the higher level, the workload, i.e., the subjobs of a DAG job, is executed by the reservations in a temporally and spatially isolated environment. The isolation property allows to analyse each DAG job's response time without inter-task interference. Most importantly, reservation systems can be co-scheduled with other tasks on the same set of physical processors using existing response time analyses.

We propose and discuss two reservation schemes, namely a *gang reservation system* in Section V-A and an *ordinary reservation system* in Section V-B, and provide resource provisioning rules and response time analyses. The hierarchical scheduling problem consists of two interconnected problems:

- Service provisioning of the respective *gang*-reservation or *ordinary*-reservation systems such that a DAG job can finish within the provided service.
- Verification of the schedulability of the provisioned reservation systems by any existing analyses that support the respective task models, e.g., sporadic arbitrary-deadline gang tasks or sporadic arbitrary-deadline ordinary sequential tasks.

For the remainder of this section, we assume the existence of a feasible schedule upon M identical multiprocessors for the studied reservation system model and focus on the service provisioning problem. We assume that a reservation system satisfies the following four properties regardless of the specific reservation model.

Property 1 (Parallel Service): The reservation systems release m parallel reservations such that at each time, during which the reservation system promises service, at most m reservations can provide service concurrently.

Property 2 (Association of Service): An instance of the reservation system serves exactly one DAG job of a DAG task. This means that an instance of the m parallel reservations that serve the ℓ-th job J_i^ℓ of DAG task τ_i all arrive synchronous at time

a_i^ℓ and the deadline is given by d_i^ℓ. Note that if the next DAG job arrives before the previous one is finished, a new instance (job) of reservations is released. This allows to directly deal with arbitrary-deadlines without further considerations.

Property 3 (Sustained Service): The service of a reservation is provided whenever the reservation system is scheduled, irrespective of whether there are insufficient number of pending subjobs to be served at that time.

Property 4 (Internal Dispatching): The reservation system's internal dispatching of each subjob $v \in V$ of a DAG job $G = (V, E)$ with parallel path-progression prioritization on m reservations follows List-FP in Definition 4 with only one difference: At any time t, let $m(t)$ denote the concurrently scheduled reservations then the $m(t)$ highest-priority pending subjobs are executed on the reservations and a lower-priority subjob is preempted if necessary.

Reclamation: The above properties are required in the formal response-time analysis, which is pessimistic in the sense that the processor may be spinning at times without executing any workload whenever a reservation is scheduled but no subjob can be dispatched. Reclamation mechanisms can be used by attaching any soft real-time DAG jobs or normal sequential jobs to the reservation with a lower-priority than the to-be served DAG job. The only required property is that the served DAG job can claim the service whenever a subjob is pending and service is provided.

A. Gang Reservation System

In gang scheduling, a set of threads is grouped together into a so-called gang with the additional constraint that all threads of a gang must be co-scheduled at the same time on available processors. It has been demonstrated that gang-based parallel computing can often improve the performance [15], [21], [41]. Due to its performance benefits, the gang model is supported by many parallel computing standards, e.g., *MPI*, *OpenMP*, *Open ACC*, or *GPU computing*. Motivated by the practical benefits and the conceptual fit of parallel path progressions in our approach and the gang execution model, we propose an m-Gang reservation system as follows.

Definition 9 (m-Gang Reservation System): A sporadic arbitrary-deadline m_i-gang reservation system $\mathcal{G}$ that serves a sporadic arbitrary-deadline DAG task $\tau_i := (G_i, D_i, T_i)$ is defined by the tuple $\mathcal{G}_i := (m_i, E_i, D_i, T_i)$ such that $m_i \cdot E_i$ service is provided during the arrival- and deadline interval with the gang scheduling constraint that all reservations must be co-scheduled at the same time.

Hence, the provisioning problem of $\mathcal{G}_i$ for a DAG task τ_i is to find m_i and E_i such that given the properties 1-4 and the gang scheduling constraint, each DAG job can complete within one of the m_i reservations before its absolute deadline.

Theorem 10 (Gang Reservation Provisioning): Each job J_i^ℓ of a sporadic arbitrary-deadline DAG task $\tau_i := (G_i, D_i, T_i)$ can complete its total volume C_i within its respective gang reservation instance of the m_i parallel reservations of size E_i before its absolute deadline if

$$vol(\pi_*) + \frac{vol(V_s^c(\psi))}{m_i - n + 1} \leq E_i \leq D_i \tag{18}$$

holds for any n- path collection ψ of at most m_i and the gang reservation system $\mathcal{G}_i$ is verified to be schedulable, i.e., able to provide all service before the absolute deadline.

Proof: Since in an m_i-gang all reservations provide service simultaneously, the arrival and finishing time window $[a_J, f_J]$ of any DAG job J of task τ_i can be partitioned into *busy*, *non-busy*, and *non-service* intervals in which none of the m_i gang reservations are scheduled and thus provide no service. In analogy to previous proofs, the response time is no more than the cumulative length of these sets, where the cumulative length of non-service times is upper-bounded by $D_i - E_i$ given the assumption that the m_i gang is schedulable. In consequence, if (18) holds, then

$$R_J \leq vol(\pi_*) + \frac{vol(V_s^c(\psi))}{m_i - n + 1} + D_i - E_i \leq D_i \tag{19}$$

which concludes the proof. □

Algorithm 2: Approximate Minimal Waste Gang.

Require: $\tau_i := (G_i, D_i, T_i)$, No. CPU M, vol, width w.
$\xi \leftarrow \emptyset$;
for each $m_i \in \{1, \ldots, \min\{w, M\}\}$ **do**
  $(\psi^*, n^*) \leftarrow$ call NPCA with G_i, m_i, vol;
  if $E_i \leftarrow$ using (18) with $(m_i, \psi^*, n^*) \leq D_i$ **then**
    $\xi \leftarrow \xi \cup (m_i, E_i, \psi^*, n^*)$;
return $\xi^* \in \xi$ such that ξ^* minimizes (20)

Gang-Reservation Provisioning: Finding the best provisioning for specific gang reservation systems depends on the concrete schedulability problem at hand and the other tasks that are to be co-scheduled. Hence, it may be beneficial to trade decreased budgets for increased gang size in some concrete scenarios. Determining such specific provisions is however beyond the scope of this work. Instead, in a more generic optimization attempt, we seek to find a provisioning that minimizes the unused gang service (waste), which is described as $m_i \cdot E_i - C_i$. Due to the gang restriction, increasing the number of reservations m_i beyond the number of processors M that the reservations are going to be executed upon is not possible. Moreover, increasing the gang size beyond the value w determined for the DAG can only increase the waste, since w describes the maximal inherent parallelism. Due to the constrained search space of $m_i \in \{1, \ldots, \min\{w, M\}\}$ an exhaustive search can be applied to find the values of m_i, E_i with $E_i \leq D_i$ and ψ, n that approximately minimize the waste objective

$$m_i \cdot vol(\pi_*) + m_i \cdot \frac{C_i - vol(V_s(\psi))}{m_i - n + 1} - C_i \tag{20}$$

as shown in Algorithm 2. For illustration of the algorithm, an exemplary result for the DAG in Fig. 1 with longest path volume 10, total volume 18, relative deadline 16 and $M = 3$ is calculated. Since, the bound w of the DAG is 4, only gang sizes of $m_i \in \{1, 2, 3\}$ are viable candidates. The algorithm returns a 2-gang with the 2-path collection $V_s(\psi) = \{v_1, v_7, v_5, v_6, v_2, v_3\}$ that yields a reservation budget of $10 + (18 - 14)/(2 - 2 + 1) = 14 \leq 16$ and waste of $2 \cdot 14 - 18 = 10$. Fig. 2 shows an exemplary schedule of this provisioned 2-gang system and the internal DAG job scheduling using the gang reservations.

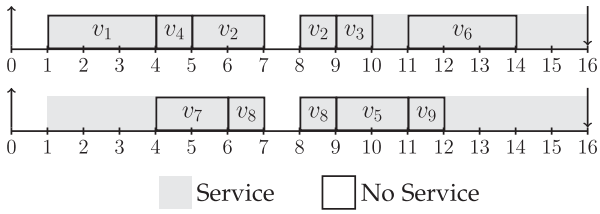


Fig. 2. Exemplary schedule for a 2-gang reservation system with a 2-path collection of the DAG illustrated in Fig. 1 with a deadline of 16 time units.

B. Ordinary Reservation System

Despite the practical benefits of gang scheduling, the analytic schedulability due to the co-scheduling constraint is reduced compared to an equivalent ordinary reservation system without such constraints.

Definition 11 (m-Ordinary Reservation System): A sporadic arbitrary-deadline m_i-ordinary reservation system $\mathcal{O}$ that serves a sporadic arbitrary-deadline DAG task $\tau_i := (G_i, D_i, T_i)$ is defined by the tuple $\mathcal{O}_i := (m_i, E_i^1, \ldots, E_i^{m_i}, D_i, T_i)$ such that $\sum_{p=1}^{m_i} E_i^p$ service is provided during the arrival- and deadline interval.

Theorem 12 (Ordinary Reservation Provisioning): Each job J_i^ℓ of a sporadic arbitrary-deadline DAG task $\tau_i := (G_i, D_i, T_i)$ can complete its total volume C_i within its respective ordinary reservation instance $\mathcal{O}_i$ before its absolute deadline if first

$$vol(\pi_*) + \frac{vol(V_s^c(\psi)) + (n-1)\cdot D_i}{m_i - n + 1} \leq \frac{\sum_{p=1}^{m_i} E_i^p}{m_i - n + 1} \quad (21)$$

holds for any n-path collection ψ where n is at most m_i and all; and second, the ordinary reservation system $\mathcal{O}_i$ is verified to be schedulable, i.e., each individual reservation is able to provide all service before the absolute deadline.

Proof: Let S be a schedule of m_i ordinary reservations that are feasibly schedulable. That is, each individual reservation is guaranteed to provide E_i^p service to the DAG job $J := J_i^\ell$ during the interval $[a_J, a_J + D_i)$ for $p \in \{1, \ldots, m_i\}$. Let ψ denote the n-path collection and $V_s(\psi)$ denote the corresponding set of subjobs. Let $m_i(t) \in \{0, 1, \ldots, m_i\}$ the number of reservations that provide service at time t and let $h(t) \in \{0, \ldots, m_i(t)\}$ the number of reservations providing service to subjobs in $V_s^c(\psi)$ at time t. We prove this theorem by contrapositive, i.e., we assume that DAG job J misses its deadline and prove that this leads to the violation of (21). Let J miss its deadline at time $d_J := a_J + D_i$ in S and let v_{k_p} denote a subjob that is executing at time d_J and is not yet finished. Please note that at least one such subjob must exist, since otherwise the total volume is finished, which contradicts the assumption of a deadline miss.

Using the envelope construction rules in Definition 5, an *incomplete envelope path* $\pi_e := \{v_{k_1}, v_{k_2}, \ldots, v_{k_p}\}$ is derived starting from subjob v_{k_p}. We partition the interval $[a_J, d_J)$ into contiguous sub intervals I_{k_i} namely $[a_{k_1}, f_{k_1}), [a_{k_2}, f_{k_2}), \ldots, [a_{k_p}, d_J)$ for each incomplete envelope subjob. Moreover, we partition each subjob interval into *busy times* defined as $\alpha_{k_i} := \{t \in I_{k_i} \mid v_{k_i} \text{ is executed}\}$ and *non-busy times* defined as $\beta_{k_i} := \{t \in I_{k_i} \mid v_{k_i} \text{ is not executed}\}$ for each $i \in \{1, \ldots, p\}$. Please note that in our partitioning, the case of no service, i.e., $m_i(t) = 0$ is considered a *non-busy time*.

To measure the cumulative amount of time spent in either state, we define

$$|\alpha_{k_i}| = \int_{I_{k_i}} [t \in \alpha_{k_i}]\, dt \text{ and } |\beta_{k_i}| = \int_{I_{k_i}} [t \in \beta_{k_i}]\, dt \quad (22)$$

where $[pred]$ denotes the iverson bracket, which evaluates to 1 if the predicate is true and 0 otherwise. Each point in time $t \in [a_J, d_J]$ is exclusively either a busy or a non-busy time and since by assumption J misses its deadline, we have that

$$D_i < \sum_{i=1}^{p} |\alpha_{k_i}| + |\beta_{k_i}| \quad (23)$$

We analyze the amount of busy times and non-busy times separately as follows.

Busy Time: The cumulative amount of *busy* times in each interval I_{k_i} is given by $|\alpha_{k_i}| \leq vol(v_{k_i})$ for $i \in \{1, \ldots, p-1\}$ and $|\alpha_{k_p}| < vol(v_{k_p})$ since the subjob v_{k_p} has not yet finished execution by definition. In summary, the cumulative amount of busy times during I_{k_i} is given by

$$\sum_{i=1}^{p} |\alpha_{k_i}| < \sum_{i=1}^{p} vol(v_{k_i}) = vol(\pi_e) \leq vol(\pi_*) \quad (24)$$

Non-Busy Time: We further partition the *non-busy* interval into a *parallel path* case if the incomplete envelope subjob $v_{k_i} \in V_s(\psi)$ and a *non-parallel path* case if $v_{k_i} \in V_s^c(\psi)$. Since our scheduling policy is work-conserving we have that whenever an envelope subjob v_{k_i} is not serviced at time $t \in I_{k_i}$ then all $m_i(t)$ reservations must be servicing non-envelope jobs (or no service is available at all).

Non-Parallel Path: Let by assumption $v_{k_i} \in V_s^c(\psi)$ then for any time $t \in \beta_{k_i}$ each of the $m_i(t)$ reservations is either exclusively servicing subjobs from $V_s^c(\psi) \setminus v_{k_i}$ (or no service is provided at all). This is due to the fact that $V_s(\psi)$ subjobs have lower-priority than $V_s^c(\psi)$ subjobs and thus the servicing of a subjob from $V_s(\psi)$ would imply the servicing of all pending $V_s^c(\psi)$ subjobs, which contradicts the assumption that pending $v_{k_i} \in V_s^c(\psi)$ is not serviced. In consequence of this implication we have that $\beta_{k_i} \subseteq \{t \in I_{k_i} \mid h(t) = m_i(t)\}$ and thus $|\beta_{k_i}|$ can be over-approximated as

$$|\beta_{k_i}| \leq \int_{I_{k_i}} [h(t) = m_i(t)]\, dt \quad (25)$$

We introduce the auxiliary function $\bar{m}_i(t) = m_i - m_i(t)$ to formalize the non-service at time t, which yields

$$|\beta_{k_i}| \leq \int_{I_{k_i}} [h(t) + \bar{m}_i(t) = m_i]\, dt \quad (26)$$

Each $t \in I_{k_i}$ that satisfies the predicate also satisfies $(h(t) + \bar{m}_i(t))/m_i = 1$. Moreover, since by definition $h(t) + \bar{m}_i(t) \geq 0$ for any $t \in I_{k_i}$ (regardless of the predicate being satisfied or not), we further approximate the length of β_{k_i} to

$$|\beta_{k_i}| \leq \int_{I_{k_i}} \frac{h(t) + \bar{m}_i(t)}{m_i}\, dt \quad (27)$$

Parallel Path: By assumption, let the incomplete envelope subjob $v_{k_i} \in V_s(\psi)$ not being serviced by any $m_i(t)$ at time $t \in I_{k_i}$. We use $n(t) \in \{1, \dots, n\}$ to denote the number of pending subjobs from $V_s(\psi)$ at time t. By case assumption, we know that for each $t \in \beta_{k_i}$ at most $n(t) - 1$ subjobs from $V_s(\psi)$ are serviced by $m_i(t)$ reservations at time t. Additionally, we know that pending $V_s^c(\psi)$ subjobs are prioritized before $V_s(\psi)$ subjobs, i.e.,

$$h(t) \geq m_i(t) - \min\{m_i(t), n(t) - 1\} \tag{28}$$

$$\geq m_i(t) - (n(t) - 1) \geq m_i(t) - (n - 1) \tag{29}$$

In consequence of this implication we have that $\beta_{k_i} \subseteq \{t \in I_{k_i} \mid h(t) \geq m_i(t) - (n-1)\}$ and thus

$$|\beta_{k_i}| \leq \int_{I_{k_i}} [h(t) \geq m_i(t) - (n-1)]\, dt \tag{30}$$

Using $\bar{m}_i(t) = m_i - m_i(t)$ yields the inequality $h(t) + \bar{m}_i(t) \geq m_i - (n-1)$. With the same reasoning as in the previous case, the length can be approximated by

$$|\beta_{k_i}| \leq \int_{I_{k_i}} \frac{h(t) + \bar{m}(t)}{m_i - (n-1)}\, dt \tag{31}$$

In conclusion we have that

$$|\beta_{k_i}| \leq \begin{cases} \text{Eq. (31)} & \text{if } v_{k_i} \in V_s(\psi) \\ \text{Eq. (27)} & \text{if } v_{k_i} \in V_s^c(\psi) \end{cases} \tag{32}$$

and since Eq. (31) $\geq$ Eq. (27), we reach the conclusion that

$$\sum_{i=1}^{p} |\beta_{k_i}| \leq \int_{a_J}^{d_J} \frac{h(t) + \bar{m}(t)}{m_i - (n-1)}\, dt \tag{33}$$

By definition, $\int_{a_J}^{d_J} h(t)\, dt \leq vol(V_s^c(\psi))$ holds and thus

$$\sum_{i=1}^{p} |\beta_{k_i}| \leq \frac{vol(V_s^c(\psi))}{m_i - n + 1} + \int_{a_J}^{d_J} \frac{\bar{m}_i(t)}{m_i - n + 1}\, dt \tag{34}$$

The contract of the reservation system for a job of DAG task τ_i promises to provide $E_i^1, \dots, E_i^{m_i}$ service during the arrival time of the DAG job J and its deadline $[a_J, d_J)$. Therefore, each of the m_i individual reservations does not provide service for at most $D_i - E_i^p$ for $p \in \{1, \dots, m_i\}$ amount of time, which implies that

$$\sum_{i=1}^{p} |\beta_{k_i}| \leq \frac{vol(V_s^c(\psi))}{m_i - n + 1} + \frac{\sum_{p=1}^{m_i} D_i - E_i^p}{m_i - n + 1} \tag{35}$$

In conclusion and with reference to (24), we have that a deadline miss of J implies that

$$D_i < vol(\pi_*) + \frac{vol(V_s^c(\psi))}{m_i - n + 1} + \frac{\sum_{p=1}^{m_i} D_i - E_i^p}{m_i - n + 1} \tag{36}$$

which proves the theorem. □

Ordinary-Reservation Provisioning Algorithm: Similar to the problem of gang reservation provisioning, trading the number of reservations with the individual reservation sizes may be beneficial for the *schedulability* of concrete task sets. Again, to provide a baseline solution that can be further refined for concrete application scenarios, we search for reservation systems that minimize the cumulative allocated service under the constraints of equal budgets $E_i^p = E_i^{p+1}$ for $p \in \{1, \dots, m_i - 1\}$ and $E_i^1 \leq D_i$, which is described in Algorithm 3. The key intuitions are; first the number of parallel reservations m_i is bounded by the minimum of the number of available processors M and the bound of the serviced DAG w, and second that the cumulative allocated service can only increase with increasing m_i. Therefore in a first stage, we set m_i to the minimal attainable value, which is n under the constraints $m_i \geq n$, resulting in the inequality

$$vol(\pi_*) + C_i - vol(V_s(\psi)) + (n-1) \cdot D_i \leq \sum_{p=1}^{n} E_i^p$$

as subject to optimization. In Algorithm 3, we compute the path-cover and bound w and apply for each $n \in \{1, \dots, \min\{w, M\}\}$ the iterative nPCA principle to search the configuration that minimizes the service by analyzing if in the n-th iteration the following improvement condition

$$-vol(V_s(\psi_n)) + (n-1) \cdot D_i < -vol(V_s(\psi_{n-1})) + (n-2) \cdot D_i$$

holds, which simplifies to $vol(V_s(\psi_n)) - vol(V_s(\psi_{n-1})) > D_i$. In case that the calculated $n^* < w$ leads to a deadline constraint violation, i.e., $E_i^* > D_i$, we iterate $m_i \in \{n^*, \dots, M\}$ until the deadline is met and return the respective solution. If $n^* = w$ then the deadline constraint can only be violated if $vol(\pi_*) > D_i$ holds, i.e., if the DAG is not schedulable by default.

Discussion: In contrast to the gang reservation system, increasing the size of the path collection in the ordinary reservation system may not be beneficial due to the additional $(n-1) \cdot D_i$ term. In order for improvements over a single path collection to

Algorithm 3: Minimal Service Ordinary Reservations.

Require: Task $\tau_i := (G_i, D_i, T_i)$, No. CPU M, vol_i.
Ensure: Minimal Service m_i-ordinary Reservations $\mathcal{O}_i$
$\psi^*, w \leftarrow \text{PATHCOVER}(G) := \{\pi_{\psi_1^*}, \dots, \pi_{\psi_w^*}\}$
$vol' \leftarrow vol$;
create $\psi_0 \leftarrow \emptyset$;
for each $n \in \{1, \dots, \min\{w, M\}\}$ **do**
  if $n = w$ **then**
    if $vol(V_s(\psi_*)) - vol(V_s(\psi_{n-1})) > D_i$ **then**
      $E_i^* \leftarrow (vol(\pi_*) + (w-1) \cdot D_i)/w$;
      return solution (E_i^*, w, ψ_*);
  else
    create $\psi_n \leftarrow \psi_{n-1}$;
    $\pi_n^* \leftarrow$ use DFS(G) to search the max. vol' path;
    $\psi_n \leftarrow \psi_n \cup \pi_n^*$;
    for each $v \in \pi_n^*$ **do**
      update $vol'(v)$ to 0;
    if $vol(V_s(\psi_n)) - vol(V_s(\psi_{n-1})) > D_i$ **then**
    $E_i^* \leftarrow (vol(\pi_*) + vol(V_s^c(\psi_n)) + (n-1)D_i)/n$;
      $n^* \leftarrow n$;
for each $m_i \in \{n^*, \dots, M\}$ **do**
  $E_i' \leftarrow ((m_i - n^*) \cdot vol(\pi_*) + n^* \cdot E_i^*)/m_i$;
  if $E_i' \leq D_i$ **then**
    return solution (E_i', m_i, ψ_n);
return infeasible;

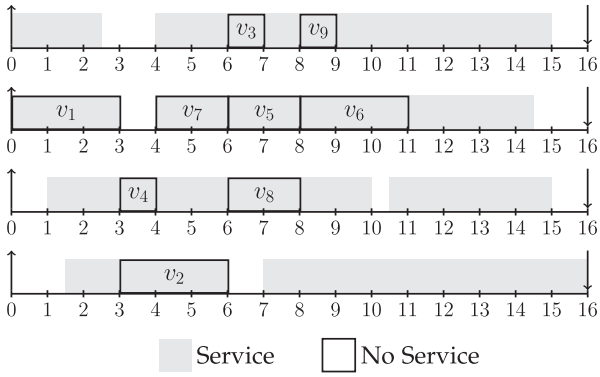


Fig. 3. Exemplary schedule for an ordinary reservation system of the DAG illustrated in Fig. 1 with a deadline of 16. A reservation system that consists of 4 equally sized reservations of 13.5 time units using a 3-path collection computed by nPCA.

be possible, the following condition must hold

$$(m-n+1)vol(\pi_*) + C_i - vol(V_s(\psi_n)) + (n-1)\cdot D_i < (m-1)vol(\pi_*) + C_i$$

which simplifies to

$$(n-1)\cdot(D_i - vol(\pi_*)) + vol(\pi_*) < vol(V_s(\psi_n)) \quad (37)$$

While the left-hand side's increase is always $D_i - vol(\pi_*)$ the right-hand side's increase is diminishing (under nPCA). Therefore, if $n = 2$ does not yield an improvement then neither does any $w > n > 2$. In general, for an n-path collection to be an improvement over the single path collection and respective reservation system, the following condition must be satisfied

$$(n-1)\cdot(D_i - vol(\pi_*)) < \sum_{k=2}^{n} vol'(\pi_k^*) \quad (38)$$

where vol' refers to the iteration based volume function in nPCA. It can be observed that very tight deadlines, i.e., $D \approx vol(\pi_*)$ and DAG structures with few overlapping paths benefit the most from this approach. In that regard the DAG used in the following example does not benefit from the parallel path concepts and only serves to illustrate the reservation principle. Note that since in Algorithm 3, the calculated cumulative allocated service no more than the cumulative allocated service of a single path reservation system, the speed up factors of $3 + 2\sqrt{2}$ under partitioned and global EDF scheduling of the arbitrary-deadline ordinary reservation system with respect to any optimal scheduler as shown by Ueter et al. [38] still apply.

Example: An exemplary 4-ordinary reservation system using a 3-path collection for the DAG shown in Fig. 1 with deadline 16 is illustrated in Fig. 3. In this example, each reservation is equal in size which results to $E_i^p = 13.5$ for $p \in \{1, \ldots, 4\}$ according to (21) with $D_i = 16$, $n = 3$, $m_i = 4$, and $vol(V_s^c(\psi)) = 4$. Please notice that the service can be provided arbitrarily depending on a concrete schedule as long as the promised service is provided within the arrival and deadline interval.

VI. Evaluation

In the forthcoming evaluations, we assess the performance of our proposed parallel path progression concepts using synthetically generated DAG task sets to allow for a systematic and exhaustive exploration. First, we evaluate the makespan of our approach (*OUR*) compared to the state-of-the-art approach as represented by He et al. [20] (*HE*). We use federated scheduling [26] (*FED*) to assess if *OUR* can leverage the the potential of parallel execution of parallel paths, since a similar performance of *OUR* and *FED* indicates that either most of the workload is on the longest path or the number of processors is insufficient, i.e., our bound degrades to Grahams bound.

Second, we evaluate our proposed gang and ordinary reservation systems provisioning strategies from Algorithm 2 (*GA*) and Algorithm 3 (*ORD*) for relative resource over-provisioning against the resource allocation of semi-federated scheduling [24] (*SEM*) and reservation-based federated scheduling [38] (*UE*) with respect to the DAG's total volume, i.e., the lower-bound.

Third, we evaluate if our path cover algorithm can outperform the iterative path collection selection in the approximation algorithm and in what parametric scenarios.

A. Experiment-Data Generation

In order to systematically evaluate the presented algorithms and to assess the performance for different classes of DAG structures, we generated 300 DAGs using the layer-by-layer and the Erdős–Rényi generation method for each configuration of generation parameters.

Layer-by-Layer Generation: The internal structure of the DAG under evaluation strongly impacts the performance of the evaluated analyses. The layer-by-layer method offers a parameterized generation process to randomly generate DAGs whose structure can be attributed to the generation parameters *min parallelism*, *max parallelism*, *min layer*, *max layers*, and *connection probability*. In each DAG's generation, the number of layers is chosen uniformly from the range $5-10$ and $10-15$ representing *min layer* to *max layer*. In each layer, the number of subtasks referred to as *parallelism* is drawn uniformly from the ranges of $5-10$, $10-15$, $10-25$, and $10-30$ representing the range of *min parallelism* to *max parallelism*. Please note that the minimal number of paths to fully cover a DAG is never more than the maximum parallelism in any of the generated layers. The connection of subtasks at a layer ℓ is only allowed by subtasks from layer $\ell - 1$. Each newly generated subtask in a layer is connected with a subtask from the previous layer with probability *connection probability*, which is drawn at random from the ranges $5\% - 10\%$, $10\% - 20\%$, $20\% - 30\%$, $40\% - 50\%$, $50\% - 60\%$, and $40\% - 80\%$ resulting in 48 different configurations for which we generated a set of 300 DAGs each. Each subtask is assigned an integer worst-case execution time drawn uniformly from the range 10 to 100.

Erdős–Rényi Generation: In addition, we generated DAG task sets using the Erdős–Rényi method that is paremeterized with *min vertex* to *max vertex* and *connection probability*. In the generation process, at first, a number of vertexes is drawn uniformly

TABLE II
DIFFERENCE OF THE PATH COVER BOUND w COMPARED TO THE MINIMAL NUMBER OF PATHS CALCULATED BY THE GREEDY APPROACH FOR A FULL COVER FOR THE LAYER-BY-LAYER DAG SETS

LAYERS	PARAL.	PROB. (%)	MAX Δ	IMPROVED (%)
5- 15	5-30	5-10	7	26
5-15	5-30	10-30	6	23
5-15	5-30	40-80	3	5

TABLE III
DIFFERENCE OF THE PATH COVER BOUND w COMPARED TO THE MINIMAL NUMBER OF PATHS CALCULATED BY THE GREEDY APPROACH FOR A FULL COVER FOR THE ERDŐS–RÉNYI DAG SETS

NO. VERTEX	PROB. (%)	MAX Δ	IMPROVED (%)
10-100	5-10	9	99%
10-100	15-20	5	96%
10-100	25-30	4	80%
10-100	35-40	2	63%
10-100	45-50	2	51%
100-150	5-10	6	75%
100-150	15-20	5	77%
100-150	25-30	3	66%
100-150	35-40	2	56%
100-150	45-50	2	38%

in the ranges of $10-100$ and $100-150$. For each class of connection probabilities $5\%-10\%$, $15\%-20\%$, $25\%-30\%$, $35\%-40\%$, and $45\%-50\%$, a *connection probability* is drawn uniformly that is used for the generation of a single DAG belonging to that class. Second, an upper-triagonal adjacency matrix is generated where each entry a_{ij} in the matrix, i.e., the directed edge (v_i, v_j), is drawn uniformly with the probability *connection probability*. We generated 300 DAGs for each combination of configurations.

Deadline and Period Generation: For each of the generated DAG sets described before, we generated *easy*, *medium*, and *hard* to schedule deadlines. That is, the open interval of deadlines $D := (vol(\pi_*), C)$ defines deadlines such that the DAG is not infeasible by default or trivially schedulable. The interval is then partitioned into three equi-sized intervals D_1, D_2 and D_3 representing the first, second, and third fraction of the interval. A deadline is considered *hard* if it is drawn uniform at random from the interval D_1, *medium* if it is drawn uniform at random from D_2, and *easy* if it is drawn uniform at random from D_3 respectively. Since the compared to methods only support constrained-deadlines except for reservation-based federated scheduling, which is a special case of our ordinary reservation system, we only evaluated constrained deadlines. We draw $\alpha \in [a, b] \subset [1, b]$ for $[a, b]$ in $\{[1, 1.2], [1.2, 1.5], [1, 2], [2, 3], [1, 1.8], [1, 3]\}$ and set the period $T = \alpha \cdot D$.

B. Path Cover Experiments

In the path cover experiment, we compare the calculated bound by the path cover algorithm that is described in Section IV-A against the minimal number of paths required by the greedy approach – described in Section IV-C – to completely cover the DAG. The motivation for this experiment is to show that there are actually cases for the evaluated DAGs in which this bound is better than the iterative solution and in consequence, the resource allocation can be significantly improved. Both algorithms are evaluated on the DAGs sets described in Section VI-A for which the results are aggregated and shown in Tables II and III respectively. The column *improved* shows the percentage of DAGs in the evaluated set for which the path cover determines less paths that the iterative solution. The column *max* shows the maximal absolute difference of the required paths, i.e., how many paths, the path cover algorithm requires less.

It can be observed that for the Erdős–Rényi set all DAG sets can be significantly improved and that sets in the range of $5-40\%$ connection probability have at least 56% improvements. For the layer-by-layer DAG sets, the improvements are still existent however only roughly a quarter of DAGs could benefit and higher connection probabilities reduce the improvements.

C. Makespan Experiment

We evaluate the makespan, i.e., the worst-case response time of a single DAG job on $4, 8, 16, 32$ processors exclusively for all configurations described in Section VI-A and present the makespan normalized to a theoretical lower-bound of $\max\{C/M, vol(\pi_*)\}$, i.e., 100% implies a tight result. Since the evaluations showed similar results, only a few representative figures are shown in the box plots in Figs. 4 and 5.

Observations: From all recorded results, it can observed that our method does not strictly dominate the approach by *HE*, but can improve the makespan in case of high parallelism, i.e., large number of processors. Intuitively, the makespan of our approach is better if the majority of the workload of the DAG task is distributed on at most M paths, which *likely* increases with the number of available processors. Therefore, the improvements depend on the DAG parameters and number of processors. Otherwise, the approach by *HE* is better in analyzing the subtask interference more accurately and thus able to provide a better makespan. Notably, our approach is however able to provide tight results for many of the evaluated cases. A representative case is shown in Fig. 4, where a DAG set with 100-150 vertexes and $45-50\%$ connection probability is evaluated. It can be observed that *OUR* provides a tight results when the number of provided processors is 8, whereas *HE* provides slightly larger (non-optimal) makespan values. Another representative case is shown in Fig. 5 for a DAG set generated by the layer-by-layer method with $10-15$ layers, a parallelism of $10-30$, and a connection probability of $50-60\%$. Note that, federated scheduling (*FED*) coincides with our analysis if only one path is considered. It can be seen that *OUR* does not significantly improve *FED* up to 16 processors, which suggests that the the majority of the workload of the DAG task is distributed on more paths. However, when 32 processors are provided for the tasks with *parallelism* of at most 30 then the makespan of *OUR* is tight in most cases.

D. Reservations Over-Provisioning Experiment

In our hierarchical scheduling approach, the schedulability depends on the schedulability analysis used for the reservation

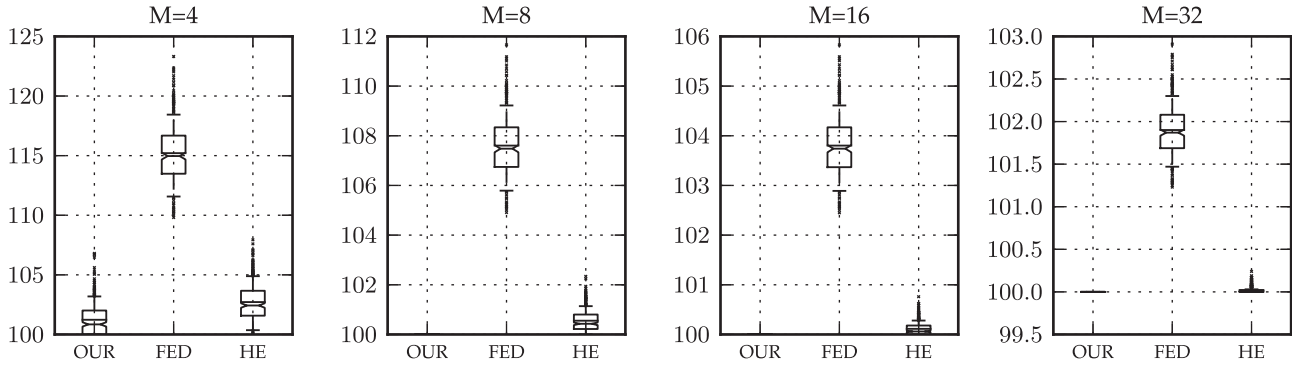


Fig. 4. Relative makespan of DAGs generated by the Erdős–Rényi method with $100-150$ vertexes and a connection probability of $45-50\%$.

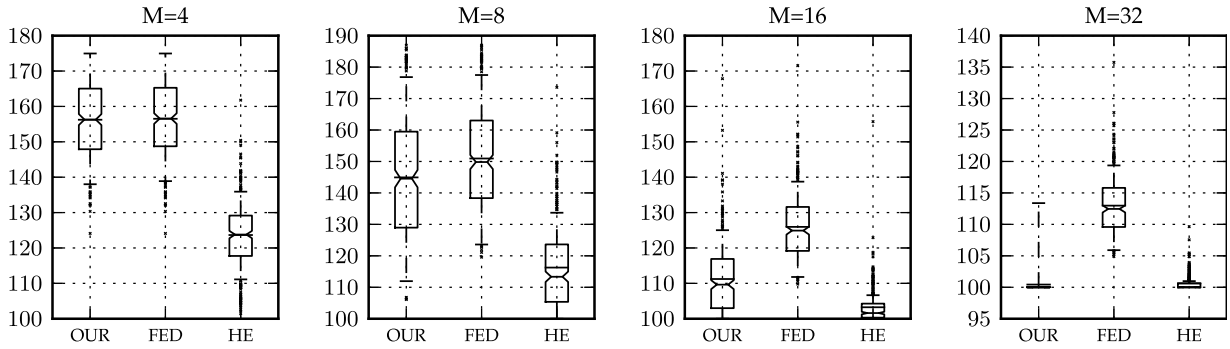


Fig. 5. Relative makespan of DAGs generated by the layer-by-layer method with $10-15$ layers, parallelism of $10-30$, and a connection probability of $50-60\%$.

systems. We are only interested in the evaluation of the resource allocation of the reservations systems as the schedulability depends on the performance of the schedulability analyses of the scheduling algorithms used to schedule the reservation systems. We compare the resource allocation of each approach per job activation (meaning over a period T) of our ordinary reservation (ORD), our gang (GA), semi-federated scheduling (SEM), and reservation-based federated scheduling (UE) relative to the DAGs total volume. That is, an over-provision value of 100% indicates a tight allocation. We assume that a sufficient number of processors is available such that a reservation system can be found for every generated deadline. A few representative results are shown in the box plots in Figs. 6 and 7.

Observation: It can be seen that *ORD* and *GA* improve the resource allocation in all scenarios and significantly for hard to scheduled DAG tasks. With increasing period to deadline ratio, the larger the improvements of the reservation based scheduling approaches to semi-federated scheduling are. Interestingly, *ORD* and *GA* show similar resource allocation in all scenarios, which demonstrates that the less restrictive reservation scheme of *ORD* does not incur larger resource demands.

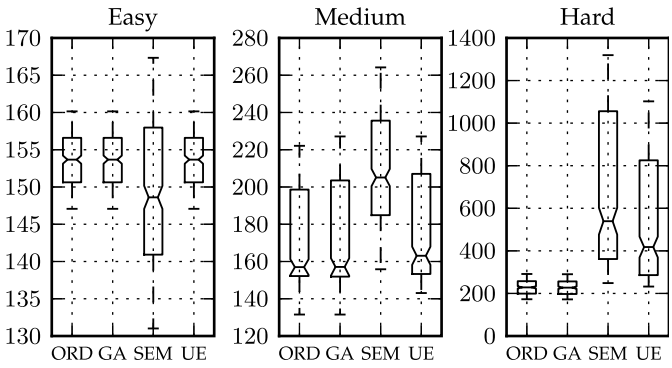


Fig. 6. Over-provisioning for Erdős–Rényi with $100-150$ vertexes, $35-40\%$ connection probability and $\alpha = [1.2, 1.5]$.

VII. Related Work

Real-time aware scheduling of parallel task systems has been extensively studied for a variety of different proposed task models. Goosens et al. [18] have provided a classification of parallel task with real-time constraints.

Early work on parallel task models focused on synchronous parallel task models, e.g., [8], [27], [33]. Synchronous parallel task models extend the fork-join model [9] in such a way that they allow different numbers of subtasks in each (synchronized) segment where the number of subtasks can exceed the number of processors.

A prominent parallel task model that has been subject to many recent scheduling and analysis efforts, is the directed-acyclic graph (DAG) task model. The DAG models subtask-level parallelism by acyclic precedence constraints for a set of subtasks. The parallel DAG task model has been studied for global [5], [7], [30] and partitioned scheduling [4], [6], [16], [17].

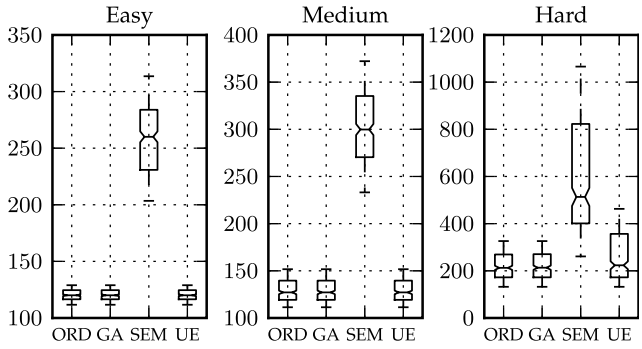


Fig. 7. Over-provisioning for layer-by-layer $10-15$ layers, parallelism $10-30\%$, connection probability $50-60\%$ and $\alpha = [2, 3]$.

The proposed scheduling algorithms and analyses in the literature can be categorized into decompositional and non-decompositional. In the former, the parallel task model is decomposed into a set of sequential task models, which is scheduled and analyzed in their stead, e.g., [23]. Non-decompositional approaches consider the peculiarities of the parallel task models, e.g., [2], [5], [13], [26], [30], [38].

A prominent decomposition based approach is federated scheduling by Li et al. [26] that avoids inter-task interference for parallel tasks. It has been extended in, e.g., [2], [3], [12], [22], [24], [38]. In the original federated scheduling approach, the set of DAG tasks is partitioned into tasks that can be executed sequentially on a single processor and tasks that need to execute in parallel to finish before their respective deadlines. In federated scheduling [26], the intra-task interference of the envelope execution is upper-bounded by the workload of the non-envelope subjobs divided by the number of processors. The corresponding response time analysis requires no information about the internal structure of the DAG except for the total volume and the longest path.

This analysis was improved by He et al. [19], who proposed a specific intra-node priority assignment for list-scheduling that the topological ordering of the nodes within the DAG. This priority assignment and the inspection of the DAG structure results in a less pessimistic upper-bound for a task's self-interference of the envelope path compared to federated scheduling. These results are further improved and extended by Zhao et al. [42], where subjob dependencies are explicitly considered along the execution of the envelope path to more accurately bound self-interference. Most recently, He et al. [20] improved their prior work by lifting the topological order restrictions in their intra-node priority assignments, which further improved the results by Zhao et al. [42].

VIII. Conclusion and Future Work

We present the parallel path progression concept that allows to analyze the simultaneous progress of a collection of parallel paths. We propose a sustainable scheduling algorithm and analysis that is extended by virtue of hierarchical scheduling for gang-based and ordinary reservation systems for sporadic arbitrary-deadline DAG tasks. For these reservations, we provide algorithms that approximately provision optimal reservation systems with respect to the service they require. We evaluated our approach using synthetically generated DAG task sets and demonstrated that our approach can improve the state-of-the-art in high-parallelism scenarios while demonstrating reasonable performance for low-parallelism scenarios. In future work, we plan to improve the active idling issues of the proposed reservation systems by considering self-suspension behaviour for the reservation systems.

References

[1] M. Andreozzi, G. Gabrielli, B. Venu, and G. Travaglini, "Industrial challenge 2022: A high-performance real-time case study on arm," in *Proc. 34th Euromicro Conf. Real-Time Syst.*, M. Maggio, Ed., Dagstuhl, Germany, 2022, pp. 1:1–1:15.

[2] S. Baruah, "The federated scheduling of constrained-deadline sporadic DAG task systems," in *Proc. Des. Automat. Test Europe Conf. Exhib.*, 2015, pp. 1323–1328.

[3] S. Baruah, "Federated scheduling of sporadic DAG task systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 179–186.

[4] S. Ben-Amor, L. Cucu-Crosjean, and D. Maxim, "Worst-case response time analysis for partitioned fixed-priority DAG tasks on identical processors," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2019, pp. 1423–1426.

[5] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *Proc. 25th Euromicro Conf. Real-Time Syst.*, 2013, pp. 225–233.

[6] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 421–433.

[7] J.-J. Chen and K. Agrawal, "Capacity augmentation bounds for parallel DAG tasks under G-EDF and G-RM," Faculty for Informatik, TU Dortmund, Dortmund Germany, Tech. Rep. 845, 2014.

[8] H. S. Chwa, J. Lee, K. Phan, A. Easwaran, and I. Shin, "Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms," in *Proc. Euromicro Conf. Real-Time Syst.*, 2013, pp. 25–34.

[9] M. E. Conway, "A multiprocessor system design," in *Proc. November 12–14, 1963, Fall Joint Comput. Conf.*, 1963, pp. 139–146.

[10] R. W. Deming, "Independence numbers of graphs-an extension of the Koenig-Egervary theorem," *Discrete Math.*, vol. 27, no. 1, pp. 23–33, 1979.

[11] R. P. Dilworth, *A Decomposition Theorem for Partially Ordered Sets*. Boston, MA, USA: Birkhäuser, 1990, pp. 7–12.

[12] S. Dinh, C. Gill, and K. Agrawal, "Efficient deterministic federated scheduling for parallel real-time tasks," in *Proc. IEEE 26th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2020, pp. 1–10.

[13] Z. Dong and C. Liu, "Work-in-progress: New analysis techniques for supporting hard real-time sporadic DAG task systems on multiprocessors," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 151–154.

[14] A. Duran et al., "OmpSs: A proposal for programming heterogeneous multi-core architectures," *Parallel Process. Lett.*, vol. 21, no. 2, pp. 173–193, 2011.

[15] D. G. Feitelson and L. Rudolph, "Gang scheduling performance benefits for fine-grain synchronization," *J. Parallel Distrib. Comput.*, vol. 16, pp. 306–318, 1992.

[16] J. Fonseca, G. Nelissen, and V. Nélis, "Improved response time analysis of sporadic DAG tasks for global FP scheduling," in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, 2017, pp. 28–37.

[17] J. C. Fonseca, G. Nelissen, V. Nélis, and L. M. Pinho, "Response time analysis of sporadic DAG tasks under partitioned scheduling," in *Proc. IEEE 11th Symp. Ind. Embedded Syst.*, 2016, pp. 290–299.

[18] J. Goossens and V. Berten, "Gang FTP scheduling of periodic and parallel rigid real-time tasks," 2010, *arXiv:1006.2617*.

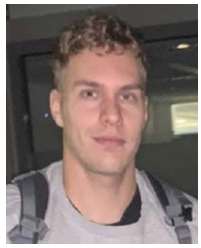
[19] Q. He, X. Jiang, N. Guan, and Z. Guo, "Intra-task priority assignment in real-time scheduling of DAG tasks on multi-cores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2283–2295, Oct. 2019.

[20] Q. He, M. Lv, and N. Guan, "Response time bounds for DAG tasks with arbitrary intra-task priority assignment," in *Proc. 33rd Euromicro Conf. Real-Time Syst.*, 2021, pp. 8:1–8:21.

[21] M. A. Jette, "Performance characteristics of gang scheduling in multiprogrammed environments," in *Proc. ACM/IEEE Conf. Supercomput.*, 1997, pp. 54–54.

[22] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and W. Yi, "Virtually-federated scheduling of parallel real-time tasks," in *Proc. IEEE Real-Time Syst. Symp.*, 2021, pp. 482–494.

[23] X. Jiang, N. Guan, X. Long, and H. Wan, "Decomposition-based real-time scheduling of parallel tasks on multicores platforms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2319–2332, Oct. 2020.
[24] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *Proc. IEEE 38th Real-Time Syst. Symp.*, 2017, pp. 80–91.
[25] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *Proc. IEEE 31st Real-Time Syst. Symp.*, 2010, pp. 259–268.
[26] J. Li, J.-J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, 2014, pp. 85–96.
[27] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-time analysis of synchronous parallel tasks in multiprocessor systems," in *Proc. 22nd Int. Conf. Real-Time Netw. Syst.*, M. Jan, B. B. Hedia, J. Goossens, and C. Maiza, Eds., 2014, Art. no. 3.
[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 *Proc. 27th Euromicro Conf. Real-Time Syst.*, 2015, pp. 211–221.
[29] A. Melani, M. A. Serrano, M. Bertogna, I. Cerutti, E. Quiñones, and G. Buttazzo, "A static scheduling approach to enable safety-critical OpenMO applications," in *Proc. 22nd Asia South Pacific Des. Automat. Conf.*, 2017, pp. 659–665.
[30] M. Nasri, G. Nelissen, and B. B. Brandenburg, "Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling," in *Proc. 31st Euromicro Conf. Real-Time Syst.*, 2019, pp. 21:1–21:23.
[31] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Math. Program.*, vol. 14, pp. 265–294, 1978.
[32] L. M. Pinho et al., "P-socrates: A parallel software framework for time-critical many-core systems," in *Proc. 17th Euromicro Conf. Digit. Syst. Des.*, 2014, pp. 214–221.
[33] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *Proc. IEEE 32nd Real-Time Syst. Symp.*, 2011, pp. 217–226.
[34] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quiñones, "Timing characterization of OpenMP4 tasking model," in *Proc. Int. Conf. Compilers Architecture Synth. Embedded Syst.*, 2015, pp. 157–166.
[35] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quiñones, "Timing characterization of OpenMP4 tasking model," in *Proc. Int. Conf. Compilers Architecture Synth. Embedded Syst.*, 2015, pp. 157–166.
[36] M. A. Serrano, S. Royuela, and E. Quiñones, "Towards an OpenMP specification for critical real-time systems," in *Proc. Int. Workshop OpenMP*, B. R. de Supinski, P. Valero-Lara, X. Martorell, S. Mateo Bellido, and J. Labarta, Eds., Cham, Springer, 2018, pp. 143–159.
[37] J. Sun, N. Guan, Y. Wang, Q. He, and W. Yi, "Real-time scheduling and analysis of OpenMP task systems with tied tasks," in *Proc. IEEE Real-Time Syst. Symp.*, 2017, pp. 92–103.
[38] N. Ueter, G. von der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 482–494.
[39] R. Vargas, E. Quinones, and A. Marongiu, "OpenMP and timing predictability: A possible union?," in *Proc. Des. Automat. Test Europe Conf. Exhib.*, 2015, pp. 617–620.
[40] R. E. Vargas, S. Royuela, M. A. Serrano, X. Martorell, and E. Quinones, "A lightweight OpenMP4 run-time for embedded systems," in *Proc. 21st Asia South Pacific Des. Automat. Conf.*, 2016, pp. 43–49.
[41] S. Wasly and R. Pellizzoni, "Bundled scheduling of parallel real-time tasks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2019, pp. 130–142.
[42] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 128–140.



Niklas Ueter (Student Member, IEEE) received the master's degree in computer science from TU Dortmund University, Germany, in 2018, and is currently working toward the PhD degree with TU Dortmund University, supervised by Prof. Dr. Jian-Jia Chen. His research interests include the area of embedded and real-time systems with a focus on real-time scheduling.

Mario Günzel (Student Member, IEEE) received the MSc degree from the Faculty of Mathematics, University of Duisburg-Essen, Germany, in 2019, and is currently working toward the PhD degree with the chair for Design Automation of Embedded Systems, TU Dortmund University, Germany, where he is supervised by Prof. Dr. Jian-Jia Chen. His research interest include the area of embedded and real-time systems. Currently, he focuses his research on the schedulability analysis of self-suspending task sets, and on end-to-end timing analysis.

Georg von der Brüggen (Senior Member, IEEE) received the diploma in computer science from TU Dortmund, in 2013, and the graduate degree from the Design Automation for Embedded Systems Group, TU Dortmund, in 2019, supervised by Jian-Jia Chen. He is a senior researcher with the Design Automation for Embedded Systems Group, TU Dortmund University, Germany, since October 2021. Beforehand, he was a postdoctoral researcher with the Max Planck Institute for Software Systems (MPI-SWS) in Kaiserslautern, Germany, as part of the Real-Time Systems Research Group headed by Björn Brandenburg since February 2020. His research interests include real-time systems and embedded systems, with a focus on real-time scheduling theory, uncertain execution behavior, dynamic real-time systems, mixed-criticality systems, probabilistic schedulability, self-suspension, multiprocessor resource sharing, utilization bounds, and speedup factors. He is regularly involved in the technical program committees of renowned international real-time conferences. He has contributed to more than 40 publications, 16 of them published in the premier conferences of the field, and received the RTSS Best Paper Award in 2021 and Outstanding Paper Awards in RTSS 2018 and RTCSA 2018.

Jian-Jia Chen (Senior Member, IEEE) received the BS degree from the Department of Chemistry, National Taiwan University, in 2001, and the PhD degree from the Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, in 2006. He is professor with the Department of Informatics, TU Dortmund University in Germany. He was junior professor with the Department of Informatics, Karlsruhe Institute of Technology (KIT) in Germany from May 2010 to March 2014. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received the European Research Council (ERC) Consolidator Award in 2019. He has received more than 10 Best Paper Awards and Outstanding Paper Awards and has involved in Technical Committees in many international conferences.