Type-Aware Federated Scheduling for Typed DAG Tasks on Heterogeneous Multicore Platforms

Ching-Chi Lin, *Member, IEEE*, Junjie Shi, *Student Member, IEEE*, Niklas Ueter, Mario Günzel, *Student Member, IEEE*, Jan Reineke, and Jian-Jia Chen, *Senior Member, IEEE*

Abstract—To utilize the performance benefits of heterogeneous multicore platforms in real-time systems, we need task models that expose the parallelism and heterogeneity of the workload, such as typed DAG tasks, as well as scheduling algorithms that effectively exploit this information. In this article, we introduce *type-aware federated scheduling* algorithms for sporadic typed DAG tasks with implicit deadlines running on a heterogeneous multicore platform with two different types of cores. In type-aware federated scheduling, a task can be executed in one of the three strategies: *Exclusive Allocation*, *Semi-Exclusive Allocation*, and *Sequential and Share*. In *Exclusive Allocation*, clusters of cores of both core types are exclusively allocated to tasks, while cores of only one type are exclusively allocated to tasks in *Semi-Exclusive Allocation*. The workload of the other type from tasks in *Semi-Exclusive Allocation* and the workload from tasks in *Sequential and Share* share the cores that are not exclusively allocated to any task. We prove that our type-aware federated scheduling algorithm has a capacity augmentation bound of 7.25. We also show that no constant capacity augmentation bound can be obtained without *Semi-Exclusive Allocation*. Compared to the state of the art, the type-aware federated scheduling algorithm achieves better schedulability, especially for task sets with skewed workload.

Index Terms—Heterogeneous multicore platforms, parallel tasks, DAG, federated scheduling, capacity argumentation bound

1 Introduction

The development of heterogeneous multicore platforms has been thriving in recent years. A heterogeneous multicore platform consists of multiple types of execution units, each with different performance and energy characteristics. Heterogeneous platforms aim to provide higher performance and more energy efficiency compared with homogeneous platforms. One concrete example for heterogeneous computing systems is the integration of main processing units with accelerators. For example, NVIDIA Tegra [19] and Samsung Exynos [21] SoCs integrate ARM processors with GPUs, while Xilinx Versal [24] integrate processors with AI accelerators on one chip.

To fully utilize the potential of a heterogeneous multicore system, we can analyze the tasks and determine the appropriate execution unit for running each code segment. Typed directed acyclic graphs (typed DAGs) commonly represent for modeling parallel real-time tasks running on heterogeneous multicore platforms. In a typed DAG task, each vertex represents a code segment that must be executed sequentially on a particular type of execution unit. Fig. 1 shows an example of a typed DAG task.

Scheduling typed DAG tasks with real-time constraints on heterogeneous multicore platforms is an emerging research topic. Most of the prior work [3], [9], [16], [18] focuses on scheduling *un-typed* real-time DAG tasks on heterogeneous platforms, and proposes methods for determining the core type each code segment (i.e., each vertex in a DAG) should be executed on. For typed DAG tasks, Han et al. [13] analyze the worst-case response time (WCRT) of a typed DAG task running on heterogeneous multicore platforms, and propose WCRT bounds with self sustainability [2]. In their follow-up paper [14], they propose a federated scheduling algorithm [17] for typed DAG tasks running on heterogeneous multicore platforms.

We note that the state of the art for federated scheduling of typed DAG tasks considers only two execution modes, i.e., *heavy* and *light*, independently of the heterogeneity of the workload distribution. However, we prove that no federated scheduling approach with only two execution modes (like that in [14]) may yield a constant capacity augmentation bound as soon as tasks with a density greater than 1 are

- *Ching-Chi Lin, Junjie Shi, Niklas Ueter, Mario Günzel, and Jian-Jia Chen are with the Design Automation for Embedded Systems Group, TU Dortmund University, 44227 Dortmund, Germany. E-mail: {chingchi.lin, junjie.shi, niklas.ueter, mario.guenzel, jian-jia.chen}@tu-dortmund.de.*
- *Jan Reineke is with Real-Time and Embedded Systems Lab, Saarland University, 66123 Saarbrücken, Germany. E-mail: reineke@cs.uni-saarland.de.*

Manuscript received 2 March 2022; revised 21 July 2022; accepted 20 August 2022. Date of publication 29 August 2022; date of current version 7 April 2023.
This work was supported in part by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under Grants 865170 and 101020415, in part by Deutsche Forschungsgemeinschaft (DFG) Sus-Aware under Grant 398602212, in part by the Federal Ministry of Education and Research (BMBF) in the course of the project 6GEM *under Grant 16KISK038, and in part by Collaborative Research Center SFB 876, subproject A3 under Grant 124020371, http://sfb876.tu-dortmund.de/)*
(Corresponding author: Ching-Chi Lin.)
Recommended for acceptance by C. Silvano.
This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2022.3202748, provided by the authors.
Digital Object Identifier no. 10.1109/TC.2022.3202748

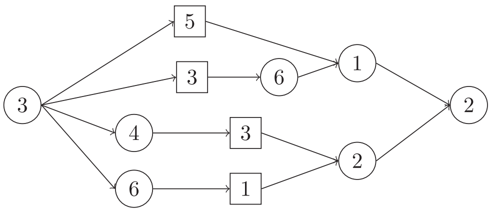


Fig. 1. Example of a typed DAG task with two types of vertices. Each circular and rectangular vertex represents a code segment that must be executed sequentially on an execution unit of the corresponding type. The number in a vertex indicates the WCET of the vertex.

classified as *heavy*, in Section 3. *Capacity augmentation bounds* [17] are one of the standard metrics to quantify the performance of scheduling algorithms for real-time systems.

In this paper, we introduce a type-aware federated scheduling algorithm for scheduling sporadic typed DAG tasks with implicit deadlines on a heterogeneous multicore platform with two types of cores. In our type-aware federated scheduling, each task is executed following one of three strategies:

1) *Exclusive Allocation:* a cluster of cores consisting of both core types is exclusively allocated to the task.
2) *Semi-Exclusive Allocation:* a cluster of cores consisting of one core type is exclusively allocated to the task. Workload of the other type is scheduled sequentially on a single core shared with other tasks.
3) *Sequential and Share:* both types of workload in the task are scheduled with other tasks on shared cores. Workload within a task is executed sequentially.

The formal definition of each execution strategy is given in Section 4. We explain how tasks are scheduled, and analyze their corresponding schedulability in Section 5. We then prove that our type-aware federated scheduling algorithm has a capacity augmentation bound of 7.25 in Section 6.

In the type-aware federated scheduling algorithm developed in Section 5, we adopt several rigid "enforcement rules" [8] to simplify the structure of the scheduling problem, and to allow the derivation of a capacity augmentation bound. Specifically, purely based on the parameters of a task, these enforcement rules determine the number of cores exclusively allocated to tasks in *Exclusive Allocation* and *Semi-Exclusive Allocation* strategies in Section 5.2. While such enforcement rules often yield constant capacity augmentation bounds, as reported by Chen et al. [8], they may also harm performance in practice by unnecessarily constraining scheduling.

Thus, in Section 7, we go on to explore an improved algorithm with the same capacity augmentation bound but without employing explicit enforcement rules. The improved algorithm is based on four principles: 1.) a sequence of attempts is made to determine the most appropriate execution strategy instead of a greedy decision based solely on the parameters of a task, 2.) preference is given to sharing over exclusive allocation where possible, 3.) the number of exclusively allocated cores is minimized for *Semi-Exclusive Allocation*, and 4.) combinatorial optimization is applied for *Exclusive Allocation*. By scheduling a task set with fewer dedicated cores, the improved type-aware federated scheduling algorithm can achieve higher schedulability in practice without sacrificing the augmentation bound.

To summarize, the contributions of this paper are as follows:

- We design a type-aware federated scheduling algorithm for scheduling sporadic typed DAG tasks with implicit deadlines on a heterogeneous multicore platform with two core types in Section 5.
- We prove a capacity augmentation bound of 7.25 for our type-aware federated algorithm in Section 6.
- We improve the type-aware federated scheduling algorithm by eliminating enforcement rules in Section 7. The improved algorithm maintains a capacity augmentation bound of 7.25 and is shown to exhibit better performance in our experimental evaluation.
- The evaluation results show that our type-aware federated scheduling algorithms achieve better schedulability on synthetic workload compared to the state of the art, especially for task sets with a skewed workload.

2 System Model and Analysis Background

In this paper, we focus on a heterogeneous multicore platform with *two* different types of execution units, i.e., cores. Let $\Theta = \{a, b\}$ be the set of core types. Existing heterogeneous computing systems such as NVIDIA Tegra [19] and Samsung Exynos [21] integrate two types of execution units, i.e., CPUs and GPUs, on one chip.

We present the task model in Section 2.1 and the problem formulation in Section 2.2. Capacity augmentation bounds are defined in Section 2.3, followed by a summary of suspension-aware schedulability analysis in Section 2.4, which we rely on in our new type-aware federated scheduling algorithm.

2.1 Typed DAG Task

A typed sporadic real-time DAG task τ_i is a 5-tuple $\tau_i = (G_i = (\mathbb{V}_i, \mathbb{E}_i), \gamma_i, \omega_i, T_i, D_i)$, where

- $\mathbb{V}_i$ is a set of vertices, in which a vertex corresponds to a piece of code that must be executed sequentially.
- $\mathbb{E}_i \subseteq \mathbb{V}_i \times \mathbb{V}_i$ is the set of directed edges in G_i. A directed edge (u, v) in $\mathbb{E}_i$ indicates a precedence constraint of the execution order of the vertices u and v in $\mathbb{V}_i$, i.e., v cannot start its execution before u finishes when $(u, v) \in \mathbb{E}_i$.
- γ_i: $\mathbb{V}_i \rightarrow \Theta$ is a function that assigns each vertex v in $\mathbb{V}_i$ to its core type. Thus, in a typed DAG task, each vertex is explicitly bound to be executed on a specific type of core.
- ω_i: $\mathbb{V}_i \rightarrow \mathbb{R}^+$ is a function that defines the WCET of each vertex v in $\mathbb{V}_i$ on its assigned core type. We assume $\omega_i(v) > 0$ for any v in $\mathbb{V}_i$.
- $T_i > 0$ is the minimum amount of time between two consecutive releases of τ_i.
- D_i is the relative deadline of τ_i. If a task is released at time r_i, all of its vertices must finish their executions no later than $r_i + D_i$.

Based on the information specified above, we can derive the WCET of a task τ_i on type a and type b cores as

$$C_i^a = \sum_{v:v\in\mathbb{V}_i\wedge\gamma_i(v)=a} \omega_i(v) \quad \text{and} \qquad C_i^b = \sum_{v:v\in\mathbb{V}_i\wedge\gamma_i(v)=b} \omega_i(v).$$

For example, in Fig. 1, $C_i^a = 24$ for type a (circular) and $C_i^b = 12$ for type b (rectangular) vertices.

We define a path π in G_i as a sequence of vertices connected via edges that starts at a *source* vertex, i.e., a vertex without predecessors, and ends at a *sink* vertex, i.e., a vertex without successors. We use $Paths(G_i)$ to denote the set of all paths in G_i. The length of a path is the sum of the vertices' WCETs on the path. The path with the longest length is called the critical path. We denote the critical path length of G_i as L_i. As the underlying graph is acyclic, L_i can be computed in linear time based on a topological ordering of the vertices. We further define L_i^a (respectively, L_i^b) to be the length of the critical path in G_i by considering only the execution times on type a (respectively, b) vertices. That is,

$$L_i^x = \max_{\pi\in Paths(G_i)} \sum_{v:v\in\pi\wedge\gamma_i(v)=x} \omega_i(v), \qquad x\in\{a,b\},$$

where $Paths(G_i)$ is the set of all paths through G_i. L_i^a (respectively, L_i^b) can be computed in the same way as L_i by temporarily setting the weights of all nodes of type b (respectively, a) to zero, and apply algorithm such as topological sorting to find the longest path in the modified DAG. By definition, $L_i^a \le L_i$, $L_i^b \le L_i$, and $L_i \le L_i^a + L_i^b$.

In this paper, we consider *implicit-deadline* task systems, in which $D_i = T_i$ for every task τ_i. The utilization of task τ_i on type a and type b is defined as $U_i^a = \frac{C_i^a}{T_i}$ and $U_i^b = \frac{C_i^b}{T_i}$, respectively. Furthermore, the worst-case response time R_i of task τ_i is an upper bound on the response time of all jobs of τ_i. Due to the assumption of implicit-deadline tasks, we have $R_i \le T_i$ for every task τ_i if τ_i meets its deadline.

2.2 Problem Formulation

We consider scheduling a set of N implicit-deadline sporadic typed DAG tasks $\mathbb{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ on a heterogeneous multicore platform with M_a type a cores and M_b type b cores, where the parameters and characteristics of each sporadic DAG task are defined in Section 2.1. Our objective is to design a scheduling algorithm that generates a task-to-core mapping and a schedule for $\mathbb{T}$, so that all jobs of the tasks in $\mathbb{T}$ finish before their deadlines. For simplicity of presentation, we implicitly assume that $C_i^a > 0$ and $C_i^b > 0$ for every task $\tau_i \in \mathbb{T}$, whilst $C_i^a = 0$ or $C_i^b = 0$ is discussed in Section B.

The following theorem shows that the problem is NP-hard in the strong sense even for special cases.

Theorem 1. *The typed DAG scheduling problem is NP-hard in the strong sense even if (1) $M_a = 1$ and $M_b = 1$, (2) $\mathbb{T}$ consists of a single task, and (3) the graph consists of chains, in which each vertex has one unit execution time.*

Proof. We show that this special case of the typed DAG scheduling problem is identical to a special case of the job shop scheduling problem. In the job shop scheduling problem, given n jobs and m machines, where a job must be processed on the machines in a given order, the objective is to minimize the makespan for completing all jobs. We consider a job shop scheduling with two shops (type a and type b), in which each shop has one machine ($M_a = 1$ and $M_b = 1$). Specifically, the scheduling problem to minimize the makespan is denoted as $J2|\text{chains}, p_{ij} = 1|C_{max}$ in three-field classification notation of scheduling problems and is NP-hard in the strong sense [23, Table 3].

Next, we construct an input instance of the studied problem, reduced from the decision version of the $J2|\text{chains}, p_{ij} = 1|C_{max}$ problem. Consider an instance with n jobs in the $J2|\text{chains}, p_{ij} = 1|C_{max}$ problem, in which D is given as the makespan constraint of the schedule. Each job must be executed on the two shops, alternating several times. Each execution in a shop takes one time unit. We can reduce this instance to an input instance of the studied problem by mapping the n jobs to one single task with n chains with a deadline D. A chain is a sequence of vertices where each vertex has only one predecessor and one successor, except for the head and tail vertex. In each chain, the operation alternates from the executions on type a and type b, each with unit execution time. Since $J2|\text{chains}, p_{ij} = 1|C_{max}$ can be reduced to the studied problem in polynomial time, we reach the conclusion. □

2.3 Capacity Augmentation Bound

The capacity augmentation bound, originally proposed in [17], is a metric for analyzing the quality of a scheduling algorithm. We first recall their definition for homogeneous multiprocessor systems. A scheduling algorithm has a capacity augmentation bound of $\frac{1}{\rho}$ $(0 < \rho \le 1)$ if any task set $\mathbb{T}$ that satisfies the following conditions is schedulable by the algorithm on M cores:

$$\sum_{\tau_i\in\mathbb{T}} U_i \le \rho M \qquad \text{and} \qquad 0 < L_i \le \rho D_i, \forall \tau_i \in \mathbb{T},$$

where L_i is the length of the critical path of task τ_i.

Since the total utilization $\sum_{\tau_i\in\mathbb{T}} U_i$ can be calculated in linear time, capacity augmentation bounds immediately yield efficient schedulability tests. Li et al. [17] proved that federated scheduling for implicit-deadline DAG sporadic tasks on homogeneous multiprocessor systems has a capacity augmentation bound of 2. They also showed that a scheduling algorithm that has capacity augmentation bound of $\frac{1}{\rho}$ also guarantees a resource augmentation bound (speed-up factor) of $\frac{1}{\rho}$.[1]

For our studied problem with two cores types, a scheduling algorithm has a capacity augmentation bound of $\frac{1}{\rho}$ $(0 < \rho \le 1)$ if any task set $\mathbb{T}$ that satisfies the following conditions is schedulable by the algorithm on M_a type a cores and M_b type b cores

$$\begin{array}{l}\sum_{\tau_i\in\mathbb{T}} U_i^a \le \rho M_a \\ \sum_{\tau_i\in\mathbb{T}} U_i^b \le \rho M_b\end{array} \qquad \text{and} \qquad 0 < L_i \le \rho D_i, \forall \tau_i \in \mathbb{T}.$$

1. Chen [4] showed that federated scheduling does not admit constant speedup bounds for constrained-deadline task systems. Therefore, we focus on implicit-deadline tasks.

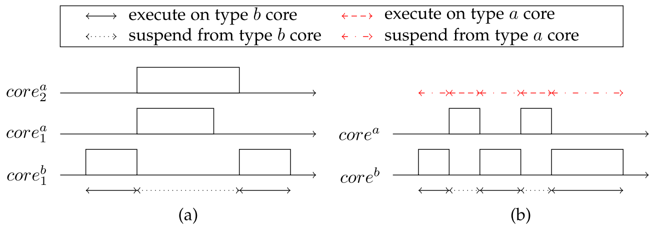


Fig. 2. Suspending behavior of a DAG task on shared cores. (a) a *Semi-Exclusive Allocation* task can be modeled as a self-suspending task running on $core_1^b$; (b) Suspending behavior of a *Sequential and Share* task from the perspectives of $core^a$ and $core^b$.

2.4 Existing Suspension-Aware Analysis

We intend to analyze the studied problem using a technique originally applied to the analysis of self-suspending tasks under preemptive static-priority scheduling on uniprocessor systems. To motivate the application of suspension-aware analysis for *uniprocessor* systems, consider the following two example settings:

1) A *Semi-Exclusive Allocation* task is assigned two exclusive cores of type a and executes its type b workload sequentially on a single type b core shared with other tasks. In our example, in Fig. 2a, $core_1^b$ is the shared type b core. From the perspective of this core, the task's executions on its exclusive cores can be modeled as suspensions, and one is left with a uniprocessor scheduling problem on $core_1^b$. The maximum suspension time can be analyzed separately, based on the number of exclusively assigned cores.
2) Similarly, a *Sequential and Share* task assigned to one shared type a and one shared type b core can be modeled as a suspending task from each core's perspective.

We now summarize an existing jitter-based suspension analysis for static-priority preemptive scheduling that we later employ in the analysis of *Semi-Exclusive Allocation* tasks. How to properly map a given *Semi-Exclusive Allocation* task to this self-suspension model is discussed later in Section 4.2.

Let τ_k be a dynamic self-suspending task with a worst-case execution time $C_k > 0$ and a maximum suspension time $S_k \geq 0$. Suppose that $hp(\tau_k)$ is the set of the higher-priority self-suspending tasks running on the same core with task τ_k. Further assume that R_i is an upper bound for the worst-case response time of τ_i with $R_i \leq T_i$ for $\tau_i \in hp(\tau_k)$. A (sufficient) schedulability test of an implicit-deadline task τ_k under static-priority preemptive scheduling due to Chen et al. [7] is

$$\exists 0 < t \leq T_k, \;\; C_k + S_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t + R_i - C_i}{T_i} \right\rceil C_i \leq t. \qquad (1)$$

We note that Chen et al. in [7] further proposed a unifying schedulability test framework, which can also be applied in our analysis without affecting our theoretical analysis. Here, we use the jitter-based analysis for simplicity of presentation. We employ Eq. (1) in Theorem 9 to validate the schedulability of a *Semi-Exclusive Allocation* task.

3 Limitation of Existing Methods

In federated scheduling, tasks are classified as *heavy* or *light* based on some metrics. For example, the state of the art proposed by Han et al. [14] classifies the tasks based on their density, i.e., the ratio of their total WCET to their deadline. In their paper, a task is *heavy* if its density is greater than 1; otherwise it is *light*. A *heavy* task is allocated to dedicated cores utilizing its DAG structure for potential parallel execution, whilst the vertices of a *light* task are *sequentially* executed on the remaining cores in competition with other *light* tasks.

We demonstrate that such federated scheduling with only two execution strategies does not yield a constant augmentation bound for scheduling typed DAG tasks on heterogeneous multi-core systems, due to tasks with *skewed* workload, i.e., heavy workload on one core type but extremely light workload on the other type. The heavy workload makes it impossible to schedule such tasks in the *light* execution strategy, while scheduling them in the *heavy* execution strategy wastes resources, which may ultimately result in non-schedulability due to an insufficient number of cores in the system.

Theorem 2. *Federated scheduling with only the light and heavy execution strategies has a capacity augmentation bound of* $\Omega(\max\{M_a, M_b\})$ *whenever a task* τ_i *with* $C_i^a + C_i^b > T_i$ *is classified as* heavy.

Proof. Consider the following example. Suppose we have a heterogeneous multi-core system consisting of $M_a > 1$ type a cores and 1 type b core. Given two fully parallel typed DAG tasks τ_1 and τ_2, i.e., there are no dependencies between vertices with different types within each task. Both of them have the same period $T \gg 1$. For τ_1, $C_1^a = \eta(T-1)$ and $C_1^b = 1$, where η is a scaling factor. For τ_2, $C_2^a = 1$ and $C_2^b = 1$.

Whenever $\eta > 1$, τ_1 is classified as a *heavy* task as $C_1^a + C_1^b > T$. As there is no available type b core anymore, it is not possible to execute the *light* task τ_2. Therefore, federated scheduling with only *heavy* and *light* execution strategies is not able to schedule both of them whenever $\eta > 1$. Since $T \gg 1$ and $\eta > 1$, we have $\frac{C_1^b + C_2^b}{T} \to 0$ and $1 < \frac{C_1^a + C_2^a}{T} < \eta$. Therefore, the capacity augmentation bound of such federated scheduling is at least $\frac{M_a}{\eta}$, which approaches M_a, when η approaches 1. □

4 Execution Modes and Allocations

The limitation of existing federated scheduling shown in Theorem 2 can be conquered by introducing a third execution strategy, *Semi-Exclusive Allocation*, described at the introduction of this paper. As there are two types of cores, this third execution strategy actually has two concrete incarnations resulting in four concrete execution modes in total.

We use a similar notation of federated scheduling in the literature to name these four execution modes (resulting from three execution strategies): *Heavy*^{ab}, *Light*, *Heavy*^{a}, and *Heavy*^{b}, in which the former two are adopted in the state of the art [14], whilst the latter two correspond to the *Semi-Exclusive Allocation* strategy introduced in this paper:

- For a task τ_i in the *Heavy*^{ab} mode, a cluster of cores consisting of m_i^a type a cores and m_i^b type b cores is exclusively allocated to a task, where m_i^a and m_i^b are positive integers. (Section 4.1)
- For a task τ_i in the *Light* mode, both types of the vertices are scheduled on *one* corresponding type of core together with other tasks. Vertices within a task are executed *sequentially*. (Section 4.3)
- For a task τ_i in the *Heavy*^{a} mode, m_i^a type a cores are exclusively allocated to task τ_i. Vertices of type b of task τ_i are *sequentially* scheduled on *one* type b core together with other tasks. (Section 4.2)
- For a task τ_i in the *Heavy*^{b} mode, m_i^b type b cores are exclusively allocated to task τ_i. Vertices of type a of task τ_i are *sequentially* scheduled on *one* type a core with other tasks. (Section 4.2)

In this section, we explain how tasks in these four modes are scheduled. How to determine which mode a task is in and how many cores are exclusively allocated to each task is discussed in Section 5.

4.1 Exclusive Allocation

A task τ_i is in the $Heavy^{ab}$ mode if m_i^a type a cores and m_i^b type b cores are *exclusively allocated* to the task. Under this scenario, there is no inter-task interference from other tasks, as only task τ_i is executed on these cores. Therefore, the schedulability of task τ_i depends only upon the internal schedule of τ_i on the m_i^a type a cores and m_i^b type b cores.

In this section, we assume that m_i^a and m_i^b are given. The details on the determination of m_i^a and m_i^b are discussed in Sections 5 and 7.

Deriving a feasible schedule to meet the timing constraint of τ_i under the specified m_i^a and m_i^b is a challenging problem. One approach is to formulate the scheduling problem as a combinatorial problem and solved with constraint programming [20], which requires high complexity to solve a problem instance. As different combinations of m_i^a and m_i^b have to be considered in our algorithm, using constraint programming is a solution with very high complexity.

Our paper adopts an alternative solution, which applies work-conserving scheduling algorithms to schedule task τ_i on the dedicated cores. The *list scheduling* algorithm has been analyzed and adopted in the literature. Specifically, list scheduling for a DAG task executed only on one core type has been widely explored in real-time systems. As an example, consider that τ_i is a DAG task with only type a workload. The analysis from Graham [11] shows that the makespan of a list schedule of a job of a DAG task on m_i^a cores is upper bounded by $L_i + (C_i^a - L(\tau_i))/m_i^a$, where L_i is the critical path length of task τ_i. If the above upper bound is no more than T_i, then the jobs of τ_i can always meet their timing constraints on the m_i^a cores assigned to τ_i exclusively.

Extending the analysis of list scheduling to a typed DAG task with multiple core types, as the problem studied in this paper, has been recently provided by Han et al. [13]. Their analysis for two core types can be summarized as follows:

Lemma 3. (Theorem 3.1 by Han et al. [13] *(rephrased)) The worst-case response time of a typed DAG task τ_i exclusively allocated on m_i^a type a cores and m_i^b type b cores for any positive integers of m_i^a and m_i^b is at most*

$$\max_{\pi\in Paths(G_i)}\left(L(\pi,a)+L(\pi,b)+\frac{C_i^a-L(\pi,a)}{m_i^a}+\frac{C_i^b-L(\pi,b)}{m_i^b}\right), \tag{2}$$

where $Paths(G_i)$ is the set of all paths in G_i, $L(\pi,a)$ and $L(\pi,b)$ are the sum of the WCETs of type a vertices and type b vertices on the path π, respectively. That is, $L(\pi,a)$ is $\sum_{v:v\in\pi\wedge\gamma_i(v)=a}\omega_i(v)$.

Proof. This comes from Theorem 3.1 in [13]. We rephrase their Eq. (4) by taking the maximum among all paths instead of defining a critical path. Moreover, Definition 3.1 in [13] defines a scaled graph in which the result is equivalent to the subtraction of the execution time (volume in their definition) by the contribution to the length on each core type. □

We can weaken the above condition in a way that is still sufficient for our capacity augmentation bound analysis. Specifically, an over-approximation of Eq. (2) can be achieved by separately considering the type a and the type b critical paths. The lengths of these paths are L_i^a and L_i^b, respectively.

Lemma 4. *The value of Eq. (2) is upper bounded by*

$$L_i^a+L_i^b+\frac{C_i^a-L_i^a}{m_i^a}+\frac{C_i^b-L_i^b}{m_i^b}. \tag{3}$$

Proof. We can rewrite Eq. (2) into $L(\pi,a)(1-\frac{1}{m_i^a})+L(\pi,b)(1-\frac{1}{m_i^b})+\frac{C_i^a}{m_i^a}+\frac{C_i^b}{m_i^b}$. By definition, $L(\pi,a)\le L_i^a$ and $L(\pi,b)\le L_i^b$ for any path $\pi\in Paths(G_i)$. We reach the conclusion as $m_i^a\ge 1$ and $m_i^b\ge 1$. □

Han et al. [13] show that the worst-case response time bound (the one we rephrased into Eq. (2)) is self-sustainable, i.e., adding more cores from one type for exclusive execution of τ_i does not make a feasible schedule of τ_i infeasible. We note that Han et al. [13] also provide a tighter analysis. However, we only need the weaker analysis here for our worst-case analysis.

4.2 Semi-Exclusive Allocation

If $C_i^a \ge T_i$ and C_i^b is positive but very small, exclusively allocating a type b core to such τ_i would be a waste, even making the task set not schedulable as demonstrated by the example in the proof of Theorem 2. In this case, it is more resource efficient to allow other tasks to also execute on the type b core that task τ_i is assigned to. In other words, τ_i should not be the only task executed on this core of type b. Interestingly, the execution behavior of task τ_i in this scenario can be modeled as a dynamic self-suspending task on a type b core, as shown in the example in Section 2.4.

Lemma 5. *Suppose that τ_i is in the* Heavy^a *mode, exclusively allocated with m_i^a type a cores. The execution behavior of task τ_i on the type b core with sequential execution can be modeled as a dynamic self-suspending task. Under list scheduling, the maximum suspension time S_i^b is at most $L_i^a+\frac{C_i^a-L_i^a}{m_i^a}$ and worst-case execution time is C_i^b under the same minimum inter-arrival time T_i.*

Symmetrically, $S_i^a = L_i^b + \frac{C_i^b - L_i^b}{m_i^b}$ *if* τ_i *is in* Heavy^b *mode.*

Proof. A task τ_i in the *Heavy*^{a} mode can be modeled as a dynamic self-suspending task running on a type b core as follows. In task τ_i, suspending from the execution on the type b core implies that only the workload of type a is executed. That is, the suspension time from the type b core is at most the amount of time executing C_i^a on the dedicated m_i^a type a cores with the critical path length of L_i^a. This is upper bounded using Lemma 4 as list scheduling is applied. By setting the worst-case execution time $C_i' = C_i^b$ and maximum suspension time $S_i' = L_i^a + \frac{C_i^a - L_i^a}{m_i^a}$, we can construct a dynamic self-suspending task τ_i' running on a type b core which is a conservative approximation of task τ_i.

The symmetric case for task in the *Heavy*^{b} mode is identical. □

4.3 Sequential Execution Without Exclusive Allocation

When both C_i^a and C_i^b are small, executing task τ_i sequentially can also be a feasible option. In this treatment, we can consider that the vertices in $\mathbb{V}_i$ are ordered by a total order (using topological sort) and executed one after another.

Definition 6. *Suppose that a typed DAG task* τ_i *is sequentially executed without any exclusive allocation on the cores, i.e.,* τ_i *is in the* Light *mode. At any time* t*, if a job of task* τ_i *is not completed yet, either*

- *the job executes or is blocked by some higher-priority job on a type* a *core (i.e., the job suspends from the type* b *core), or*
- *the job executes or is blocked by some higher-priority job on a type* b *core (i.e., the job suspends from the type* a *core).*

The execution can be modeled as a 3-tuple (C_i^a, C_i^b, T_i) *with sequential executions in an interleaving manner on type* a *and type* b *cores.* □

5 Type-Aware Federated Scheduling

In this section, we discuss the Federated Scheduling paradigm. We provide the schedulability analysis for each execution mode in Section 5.1. In Section 5.2, we describe how to determine the execution mode of a task, and how many cores are exclusively allocated to each task in the *Heavy*^{ab}, *Heavy*^{a}, and *Heavy*^{b} modes.

According to the definition, we have four task execution modes: *Heavy*^{ab}, *Heavy*^{a}, *Heavy*^{b}, and *Light*, as described in Section 4. Table 1 summarizes the four execution modes.

We start with the definition of the type-aware federated static-priority preemptive scheduling for typed DAG tasks.

Definition 7. Type-aware federated static-priority preemptive scheduling for typed DAG tasks*:*

1) *Each task is in one of the four task execution modes:* Heavy^{ab}, Heavy^a, Heavy^b, *and* Light.
2) *If task* τ_i *is in the* Heavy^{ab}, Heavy^a, *or* Heavy^b *mode, a cluster of cores with the corresponding core types are dedicated to* τ_i*. That is, each of these cores only has one task assigned to it.*
3) *For the core type without exclusive allocation of task* τ_i*, the task is assigned to be executed on* one assigned core. *When there are multiple tasks assigned to a core,* static-priority preemptive scheduling *on the core is applied.* □

TABLE 1
Summary of the Execution Modes

Mode	Type a	Type b	Schedulability Analysis
Heavy^{ab}	Exclusive allocation	Exclusive allocation	Response time analysis in [13] or constraint programming
Heavy^a	Exclusive allocation	Shared	Dynamic self-suspension task on type b core, Eq. (4) in Theorem 9
Heavy^b	Shared	Exclusive allocation	Dynamic self-suspension task on type a core, Eq. (5) in Theorem 9
Light	Shared	Shared	Sequential execution, based on [15] and restated in Theorem 10

We use rate monotonic scheduling priority assignment, i.e., a task τ_i has a higher priority than a task τ_k if $T_i \leq T_k$, in which ties are broken arbitrarily. For the rest of this paper, we use p to denote a core of type a and q to denote a core of type b. The set of tasks that are assigned to core p and core q are denoted as Ψ_p^a and Ψ_q^b, respectively. $\Psi_p^a(\tau_k)$ (respectively, $\Psi_q^b(\tau_k)$) is the set of tasks that are assigned to core p (respectively, q) that have higher priorities than τ_k. Under type-aware federated static-priority preemptive scheduling, when a task τ_i is in $\Psi_p^a(\tau_k)$ (respectively, $\Psi_q^b(\tau_k)$) and the WCET of τ_i on core p is C_i^a (respectively, on core q is C_i^b), a job of τ_i can block a single job of τ_k from execution with at most C_i^a time units on core p (respectively, C_i^b times units on core q).

5.1 Schedulability Analysis

Given the execution mode of a task, we can validate the schedulability of the task based on the following theorems.

Theorem 8. *A task in the* Heavy^{ab} *mode meets its deadline if the worst-case response time in Lemma 3 or 4 is no more than its relative deadline.*

Proof. As there is no other task assigned to the cores exclusively allocated to the task, the theorem holds naturally. □

Theorems 9 and 10 are the schedulability tests for tasks in the *Heavy*^{a}, *Heavy*^{b}, and *Light* mode. As the tests in Theorems 9 and 10 require the worst-case response time R_i of a higher-priority task τ_i, when analyzing the schedulability of τ_k, we should also set the corresponding R_k after handling τ_k accordingly. That is, R_k is set to be the minimum t satisfying the corresponding condition applied for τ_k in Eqs. (4), (5), or (6).

Theorem 9. *Suppose that* $R_i \leq T_i, \forall \tau_i \in \Psi_q^b(\tau_k)$*. Task* τ_k *in the* Heavy^a *mode assigned to core q (i.e., core type b) meets its deadline if*

$$\exists 0 < t \leq T_k, \;\; C_k^b + S_k^b + \sum_{\tau_i \in \Psi_q^b(\tau_k)} \left\lceil \frac{t + R_i - C_i^b}{T_i} \right\rceil C_i^b \leq t. \qquad (4)$$

Suppose that $R_i \leq T_i, \forall \tau_i \in \Psi_p^a(\tau_k)$. *Symmetrically,* τ_k *in the* Heavy^b *mode assigned to core* p *(i.e., core type* a*) meets its deadline if*

$$\exists 0 < t \leq T_k, \;\; C_k^a + S_k^a + \sum_{\tau_i \in \Psi_p^a(\tau_k)} \left\lceil \frac{t + R_i - C_i^a}{T_i} \right\rceil C_i^a \leq t. \quad (5)$$

Proof. According to Lemma 5, the execution behavior of a task τ_k in the *Heavy*^{a} mode (*Heavy*^{b} mode, respectively) can be modeled as a dynamic self-suspending task running on core q with type b (core p with type a, respectively). By substituting the worst-case execution time C_k and the maximum suspension time S_k with C_k^b and S_k^b (C_k^a and S_k^a, respectively) in the schedulability test in Eq. (1), we reach the conclusion. □

Theorem 10. *Suppose that* $R_i \leq T_i, \forall \tau_i \in \Psi_p^a(\tau_k) \cup \Psi_q^b(\tau_k)$. *Task* τ_k *in the* Light *mode assigned to core* p *of core type* a *and core* q *of core type* b *meets its deadline if*

$$\exists 0 < t \leq T_k, \quad \begin{pmatrix} C_k^a & + \sum_{\tau_i \in \Psi_p^a(\tau_k)} \left\lceil \frac{t+R_i-C_i^a}{T_i} \right\rceil C_i^a \\ + \;\; C_k^b & + \sum_{\tau_i \in \Psi_q^b(\tau_k)} \left\lceil \frac{t+R_i-C_i^b}{T_i} \right\rceil C_i^b \end{pmatrix} \leq t. \quad (6)$$

Proof. This comes from the symmetric view of execution on core p and on core q using Theorem 1 in [15] (stated in the Appendix), available online. In [15], Huang et al. provide a resource-centric symmetric timing analysis for real-time tasks on multi-core platforms with shared resources. We adopt their timing analysis for τ_k by considering core p as a resource shared by all $\tau' \in \Psi_p^a$. By setting B, i.e., the overhead for requesting the shared resource, in [15] to 0, the response time of τ_k is upper bounded by $X(t) + S(t)$ according to Theorem 1 in [15], where $X(t)$ is the amount of time that τ_k is accessing the shared resource, and $S(t)$ is the amount of time that τ_k is suspended from the shared resource (core p). This also corresponds to Definition 6. Since that $X(t)$ is upper bounded by $C_k^a + \sum_{\tau_i \in \Psi_p^a(\tau_k)} \left\lceil \frac{t+R_i-C_i^a}{T_i} \right\rceil C_i^a$ and $S(t)$ is upper bounded by $C_k^b + \sum_{\tau_i \in \Psi_q^b(\tau_k)} \left\lceil \frac{t+R_i-C_i^b}{T_i} \right\rceil C_i^b$, the theorem holds. □

5.2 Greedy Type-Aware Federated Scheduling Algorithm

After presenting the analysis and the scheduling philosophy, we present our scheduling algorithm based on a greedy approach. The algorithm consists of two parts.

In the first part, we classify the tasks in the input task set $\mathbb{T}$ into four classes based on a control parameter ρ, $0 < \rho \leq 0.5$:

1) Task τ_i is in the *Heavy*^{ab} mode when $C_i^a > \rho T_i$ and $C_i^b > \rho T_i$.
2) Task τ_i is in the *Heavy*^{a} mode when $C_i^a > \rho T_i$ and $C_i^b \leq \rho T_i$
3) Task τ_i is in the *Heavy*^{b} mode when $C_i^a \leq \rho T_i$ and $C_i^b > \rho T_i$
4) Task τ_i is in the *Light* mode when $C_i^a \leq \rho T_i$ and $C_i^b \leq \rho T_i$

For a given ρ, this step classifies the tasks in $\mathbb{T}$ into four disjoint sets. Here, we use $\mathbb{H}^{ab}$, $\mathbb{H}^a$, $\mathbb{H}^b$, and $\mathbb{LI}$ to denote these four sets for simplicity.

In the second part, we first handle the number of cores exclusively allocated to the tasks in $\mathbb{H}^{ab}$, $\mathbb{H}^a$, and $\mathbb{H}^b$.

1) For a task τ_i in $\mathbb{H}^{ab}$, we set $m_i^a = \left\lceil \frac{C_i^a - L_i^a}{\frac{T_i}{2} - L_i^a} \right\rceil$ and $m_i^b = \left\lceil \frac{C_i^b - L_i^b}{\frac{T_i}{2} - L_i^b} \right\rceil$.
2) For a task τ_i in $\mathbb{H}^a$, we set $m_i^a = \left\lceil \frac{C_i^a - L_i^a}{\frac{T_i}{3} - L_i^a} \right\rceil$.
3) For a task τ_i in $\mathbb{H}^b$, we set $m_i^b = \left\lceil \frac{C_i^b - L_i^b}{\frac{T_i}{3} - L_i^b} \right\rceil$.

If any of the above setting of m_i^a and/or m_i^b is negative, we return failure. Otherwise, we allocate the dedicated cores for the tasks in $\mathbb{H}^{ab}$, $\mathbb{H}^a$, and $\mathbb{H}^b$ accordingly. By enforcing the number of dedicated cores allocated to tasks with the above procedure, the derivation of the capacity augmentation bound is easier to present. *This, however, may sacrifice the schedulability.*

Let $M_a^\sharp$ be $M_a - \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} m_i^a$ and $M_b^\sharp$ be $M_b - \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} m_i^b$ as the remaining number of type a and type b cores after allocating the dedicated cores. If it is not possible to allocate enough dedicated cores, i.e., if $M_a^\sharp < 0$ or $M_b^\sharp < 0$, we return failure. Otherwise, we continue to schedule and partition the tasks in $\mathbb{H}^a$, $\mathbb{H}^b$, and $\mathbb{LI}$ to resolve the competition on the shared $M_a^\sharp$ type a cores and $M_b^\sharp$ type b cores.

We order the priorities of the tasks in $\mathbb{H}^a$, $\mathbb{H}^b$, and $\mathbb{LI}$ in the rate-monotonic manner, and start from the task with the highest priority (shortest T_i). Suppose the next task to be assigned is τ_k.

1) If τ_k is in $\mathbb{LI}$, we try to assign it on a core p of core type a and a core q of core type b, and apply the schedulability test in Theorem 10. If it is not possible under any combination, we return failure; otherwise, one combination of p and q is selected to assign task τ_k onto.
2) If τ_k is in $\mathbb{H}^a$, we try to assign it on a core q of type b, and apply the schedulability test in Theorem 9. If it is not possible, we return failure; otherwise, one q is selected to assign task τ_k onto.
3) Symmetrically, if τ_k is in $\mathbb{H}^b$, we try to assign it on a core p of type a like above.

After all tasks are feasibly assigned and scheduled, the task-to-core mapping is returned as a feasible solution.

Discussions of the parameters of the greedy algorithm

Selection of ρ: The greedy algorithm classifies the tasks into four execution modes according to the control parameter ρ. If we pick a small ρ, more tasks are in the *Heavy*^{ab} mode, which requires more exclusively allocated cores. On the other hand, a larger ρ results in more *Light* tasks. Therefore, ρ should be carefully selected, detailed in Section 6.

Fitting strategies: first fit, worst fit, best fit: For the tasks in the *Light*, *Heavy*^{a}, and *Heavy*^{b} modes, finding a core p of core type a and/or a core q of core type b can be formulated as a bin packing problem. We can apply existing fitting strategies such as First-Fit, Worst-Fit, or Best-Fit to find a core (or a pair of cores) that is schedulable for the task. For First-Fit, we assign the task to the first core (or first pair of cores in the *Light* mode) in the list that can schedule the task by following predefined indexes of cores. For Best-Fit (Worst-Fit, respectively), a task in the *Heavy*^{a} or *Heavy*^{b} mode is assigned to the feasible core that has the highest utilization (lowest utilization, respectively) after assigning the task. Furthermore, a task in the *Light* mode is assigned to the pair of feasible cores that have the highest sum of utilization (lowest sum of utilization, respectively) after assigning the task for Best-Fit (Worst-Fit, respectively). Note that we only assign the task to an unused core, i.e., a core that has not been assigned with any task yet, if the task is not schedulable on any used core.

Time complexity: We can directly compute the number of type a and type b cores that are exclusively allocated to tasks in $\mathbb{H}^{ab}$, $\mathbb{H}^a$, and $\mathbb{H}^b$. For tasks that share cores, i.e., tasks in the *Light*, *Heavy*^{a}, and *Heavy*^{b} modes, there are at most $O(M_a M_b)$ possible combinations of p and q when considering τ_k. The schedulability test in Theorems 9 and 10 and their worst-case response time analysis require $O(kT_k)$ time complexity. The time complexity of the fitting strategy to select cores for τ_k is $O(M_a M_b)$. Therefore, the overall time complexity is $O(N^2 M_a M_b T_N)$, where T_N is the maximum inter-arrival time of the given N tasks. The time complexity is pseudo-polynomial, which can be reduced by approximating the tests in Theorems 9 and 10 in polynomial time [7].

6 Capacity Augmentation Bound

We prove the capacity augmentation bound of our greedy type-aware federated scheduling algorithm in Section 5.2 in this section. First, we provide the upper bound for the total number of type a and type b cores that exclusively allocated to tasks in Theorem 13. We then prove the schedulability of the workload that share the remaining cores. The capacity augmentation bound our greedy type-aware federated scheduling algorithm is proved to be 7.25 in Theorem 17.

To prove the upper bound for the total number of type a and type b cores that exclusively allocated to tasks, we derive the following two lemmas. Recall that we enforced the number of dedicated cores allocated to tasks in the *Heavy*^{ab} mode as $m_i^a = \left\lceil \frac{C_i^a - L_i^a}{\frac{T_i}{2} - L_i^a} \right\rceil$ and $m_i^b = \left\lceil \frac{C_i^b - L_i^b}{\frac{T_i}{2} - L_i^b} \right\rceil$ in the algorithm. Similarly, $m_i^a = \left\lceil \frac{C_i^a - L_i^a}{\frac{T_i}{3} - L_i^a} \right\rceil$ and $m_i^b = \left\lceil \frac{C_i^b - L_i^b}{\frac{T_i}{3} - L_i^b} \right\rceil$ for tasks in the *Heavy*^{a} mode and *Heavy*^{b} mode, respectively.

Lemma 11. *For any $0 < \rho \leq 1/4$, when $0 < L_i^a \leq \rho T_i < C_i^a$ and $0 < L_i^b \leq \rho T_i < C_i^b$, under the greedy type-aware federated scheduling algorithm, for every task $\tau_i \in \mathbb{H}^{ab}$, we have*

$$m_i^a \leq \frac{U_i^a}{\rho} \qquad \text{and} \qquad m_i^b \leq \frac{U_i^b}{\rho}. \tag{7}$$

Proof. The algorithm sets

$$m_i^a = \left\lceil \frac{C_i^a - L_i^a}{\frac{T_i}{2} - L_i^a} \right\rceil = \left\lceil \frac{U_i^a - \frac{L_i^a}{T_i}}{\frac{1}{2} - \frac{L_i^a}{T_i}} \right\rceil.$$

If $U_i^a \geq 1/2$, since $0 < L_i^a/T_i \leq \rho \leq 1/4$, we have

$$m_i^a \leq \left\lceil \frac{U_i^a - \rho}{\frac{1}{2} - \rho} \right\rceil \leq \frac{U_i^a - \frac{1}{4}}{\frac{1}{2} - \frac{1}{4}} + 1 = 4U_i^a \leq \frac{U_i^a}{\rho}. \tag{8}$$

If $U_i^a < 1/2$, since $L_i^a/T_i^a \leq \rho < U_i^a$, we have

$$m_i^a = \left\lceil \frac{U_i^a - \frac{L_i^a}{T_i}}{\frac{1}{2} - \frac{L_i^a}{T_i}} \right\rceil = 1 \leq \frac{U_i^a}{\rho}. \tag{9}$$

The case for m_i^b is identical. □

Lemma 12. *For any $0 < \rho \leq 1/6$, when $0 < L_i^a \leq \rho T_i < C_i^a$ and $0 < L_i^b \leq \rho T_i < C_i^b$, under the greedy type-aware federated scheduling algorithm we have*

$$m_i^a \leq \frac{U_i^a}{\rho}, \forall \tau_i \in \mathbb{H}^a \quad \text{and} \quad m_i^b \leq \frac{U_i^b}{\rho}, \forall \tau_i \in \mathbb{H}^b. \tag{10}$$

Proof. For a task τ_i in $\mathbb{H}^a$, the algorithm sets

$$m_i^a = \left\lceil \frac{C_i^a - L_i^a}{\frac{T_i}{3} - L_i^a} \right\rceil = \left\lceil \frac{U_i^a - \frac{L_i^a}{T_i}}{\frac{1}{3} - \frac{L_i^a}{T_i}} \right\rceil.$$

If $U_i^a \geq 1/3$, since $0 < L_i^a/T_i \leq \rho \leq 1/6$, we have

$$\begin{aligned} m_i^a &\leq \left\lceil \frac{U_i^a - \rho}{\frac{1}{3} - \rho} \right\rceil \leq \left\lceil \frac{U_i^a - \frac{1}{6}}{\frac{1}{3} - \frac{1}{6}} \right\rceil = \lceil 6U_i^a - 1 \rceil \\ &\leq 6U_i^a - 1 + 1 = 6U_i^a \leq \frac{U_i^a}{\rho}, \end{aligned} \tag{11}$$

If $U_i^a < 1/3$, since $L_i^a/T_i^a \leq \rho < U_i^a$, we have

$$m_i^a = \left\lceil \frac{U_i^a - \frac{L_i^a}{T_i}}{\frac{1}{3} - \frac{L_i^a}{T_i}} \right\rceil = 1 \leq \frac{U_i^a}{\rho}. \tag{12}$$

The case for m_i^b is identical for task τ_i in $\mathbb{H}^b$. □

By the above two lemmas, we have the following theorem.

Theorem 13. *For any $0 < \rho \leq 1/6$, when $0 < L_i^a \leq \rho T_i$ and $0 < L_i^b \leq \rho T_i$ for every task $\tau_i \in \mathbb{T}$,*

$$\sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} m_i^a \leq \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} \frac{U_i^a}{\rho} \tag{13}$$

and

$$\sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} m_i^b \leq \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} \frac{U_i^b}{\rho}. \tag{14}$$

Proof. It comes from the combination of Lemmas 11 and 12. □

Next, we prove the schedulability of the tasks in the *Heavy*^{a}, *Heavy*^{b}, and *Light* modes on the remaining cores. We need the following lemma which is based on the generalized utilization-based analysis framework by Chen et al. [5].

Lemma 14. *Suppose that* $T_i \le T_k$ *and* $y_i > 0$ *for every* $i \in \mathbb{Y}$ *and* $x_k > 0$. *The following condition*

$$\forall 0 < t \le T_k, \;\; x_k + \sum_{i \in \mathbb{Y}} \left\lceil \frac{t + T_i}{T_i} \right\rceil y_i > t, \tag{15}$$

implies that

$$\sum_{i \in \mathbb{Y}} \frac{y_i}{T_i} > \ln\left(\frac{3}{2 + \frac{x_k}{T_k}}\right). \tag{16}$$

Proof. The proof can be achieved by following the suggested procedure in [5] by specifying the corresponding parameters. It can be found in the Appendix, available in the online supplemental material. □

Recall that $M_a^\sharp = M_a - \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} m_i^a$ and $M_b^\sharp = M_b - \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} m_i^b$ are the remaining number of type a and type b cores after allocating the dedicated cores. We have the following two lemmas.

Lemma 15. *For any* $0 < \rho \le 1/6$, *when* $0 < L_i^a \le \rho T_i$ *and* $0 < L_i^b \le \rho T_i$ *for every task* $\tau_i \in \mathbb{T}$, *if task* τ_k *in* $\mathbb{H}^a$ *is the first task that cannot be feasibly scheduled on one of the* $M_b^\sharp > 0$ *cores under the greedy type-aware federated scheduling algorithm, then*

$$\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} U_i^b > \rho M_b^\sharp. \tag{17}$$

Similarly, if task τ_k *in* $\mathbb{H}^b$ *is the first task that cannot be feasibly scheduled on one of the* $M_a^\sharp > 0$ *cores, then*

$$\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} U_i^a > \rho M_a^\sharp. \tag{18}$$

Proof. By Theorem 9, for every core q among the $M_b^\sharp$ type b cores, we have

$$\forall 0 < t \le T_k, \;\; C_k^b + S_k^b + \sum_{\tau_i \in \Psi_q^b(\tau_k)} \left\lceil \frac{t + R_i - C_i^b}{T_i} \right\rceil C_i^b > t. \tag{19}$$

Since higher-priority tasks meet their deadlines before considering τ_k, we have $R_i \le T_i$ for every task $\tau_i \in \Psi_q^b(\tau_k)$. Therefore,

$$\forall 0 < t \le T_k, \;\; C_k^b + S_k^b + \sum_{\tau_i \in \Psi_q^b(\tau_k)} \left\lceil \frac{t + T_i}{T_i} \right\rceil C_i^b > t. \tag{20}$$

Recall that m_k^a is set to $\left\lceil \frac{C_k^a - L_k^a}{\frac{T_k}{3} - L_k^a} \right\rceil$ when τ_k is in $\mathbb{H}^a$. Since $L_k^a \le \rho T_k \le T_k/6$, by Lemma 5, we have

$$S_k^b \le L_k^a + \frac{C_k^a - L_k^a}{m_k^a} \le L_k^a + \frac{C_k^a - L_k^a}{\frac{C_k^a - L_k^a}{\frac{T_k}{3} - L_k^a}} = \frac{T_k}{3}.$$

Furthermore, since $\frac{C_k^b}{T_k} \le \rho \le 1/6$, we have $\frac{C_k^b + S_k^b}{T_k} \le 1/2$. To prove a lower bound on $\sum_{\tau_i \in \Psi_q^b(\tau_k)} U_i^b$, we adopt the generalized utilization-based analysis framework stated in Lemma 14 by reformulating Eq. (20) using x_k as $C_k^b + S_k^b$ and y_i as C_i^b for every $\tau_i \in \Psi_q^b(\tau_k)$

$$\forall 0 < t \le T_k, \;\; x_k + \sum_{\tau_i \in \Psi_q^b(\tau_k)} \left\lceil \frac{t + T_i}{T_i} \right\rceil y_i > t. \tag{21}$$

By adopting Lemma 14, the condition in Eq. (21) leads to

$$\sum_{\tau_i \in \Psi_q^b(\tau_k)} U_i^b > \ln\left(\frac{3}{2 + \frac{C_k^b + S_k^b}{T_k}}\right) \ge \ln\left(\frac{3}{2.5}\right) > 1/6 \ge \rho.$$

Since every higher-priority task τ_i is dedicated to at most one core q under the type-aware federated scheduling paradigm, we know that

$$\rho M_b^\sharp < \sum_{q=1}^{M_b^\sharp} \sum_{\tau_i \in \Psi_q^b(\tau_k)} U_i^b \le \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} U_i^b.$$

The other case for $M_a^\sharp$ follows as well. □

Lemma 16. *For any* $0 < \rho \le 1/7.25$, *when* $0 < L_i^a \le \rho T_i$ *and* $0 < L_i^b \le \rho T_i$ *for every task* $\tau_i \in \mathbb{T}$, *if task* τ_k *in* $\mathbb{LI}$ *is the first task that cannot be feasibly scheduled on one of the* $M_a^\sharp > 0$ *type* a *cores together with one of the* $M_b^\sharp > 0$ *type* b *cores under the greedy type-aware federated scheduling algorithm, then*

$$\frac{\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} U_i^b}{M_b^\sharp} + \frac{\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} U_i^a}{M_a^\sharp} > 2\rho. \tag{22}$$

Thus, either Eqs. (17) or (18) holds.

Proof. By Theorem 10 and the fact $R_i \le T_i$ for every higher-priority task τ_i before considering τ_k, for every of core p among the $M_a^\sharp$ type a cores and every of core q among the $M_b^\sharp$ type b cores, we have

$$\forall 0 < t \le T_k,$$
$$C_k^a + C_k^b + \sum_{\tau_i \in \Psi_p^a(\tau_k)} \left\lceil \frac{t + T_i}{T_i} \right\rceil C_i^a + \sum_{\tau_i \in \Psi_q^b(\tau_k)} \left\lceil \frac{t + T_i}{T_i} \right\rceil C_i^b > t.$$

Therefore, for all combinations of $M_a^\sharp M_b^\sharp$ pairs of p and q, the above condition holds. By summing up all these $M_a^\sharp M_b^\sharp$ inequalities, we have

$$\forall 0 < t \le T_k,$$
$$M_a^\sharp M_b^\sharp (C_k^a + C_k^b) + M_b^\sharp \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} \left\lceil \frac{t + T_i}{T_i} \right\rceil C_i^a + M_a^\sharp \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} \left\lceil \frac{t + T_i}{T_i} \right\rceil C_i^b > M_a^\sharp M_b^\sharp t. \tag{24}$$

Dividing both sides with $M_a^\sharp M_b^\sharp$, we have

$$\begin{aligned}&\forall 0 < t \le T_k,\\ &C_k^a + C_k^b + \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} \left\lceil \frac{t+T_i}{T_i} \right\rceil \frac{C_i^a}{M_a^\sharp}\\ &+ \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} \left\lceil \frac{t+T_i}{T_i} \right\rceil \frac{C_i^b}{M_b^\sharp} > t. \end{aligned} \tag{25}$$

We again utilize Lemma 14 like the proof of Lemma 15. By setting x_k to $C_k^b + S_k^b$ and y_i to $\frac{C_i^a}{M_a^\sharp} + \frac{C_i^b}{M_b^\sharp}$ for $\tau_i \in \mathbb{LI}$, y_i to $\frac{C_i^a}{M_a^\sharp}$ for $\tau_i \in \mathbb{H}^b$, and y_i to $\frac{C_i^b}{M_b^\sharp}$ for $\tau_i \in \mathbb{H}^a$ in the above condition when adopting Lemma 14, we have

$$\begin{aligned}\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} \frac{U_i^a}{M_a^\sharp} + \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} \frac{U_i^b}{M_b^\sharp} &> \ln \frac{3}{2 + \frac{C_k^b + C_k^b}{T_k}} \ge \ln \frac{3}{2+2\rho}\\ &\ge \ln \frac{3}{2 + 2/7.25} = \frac{2}{7.2428\cdots} > \frac{2}{7.25}.\end{aligned} \tag{26}$$

□

We are ready to prove the capacity augmentation bound of the greedy type-aware federated scheduling algorithm.

Theorem 17. *The capacity augmentation bound of the greedy type-aware federated scheduling algorithm is at most 7.25. That is, when* ρ *is* $1/7.25$, *if*

$$\begin{aligned}&\max\{L_i^a, L_i^b\} \le L_i \le \rho T_i, \ \ \forall \tau_i \in \mathbb{T} \text{ and}\\ &\sum_{\tau_i \in \mathbb{T}} U_i^a \le \rho M_a \text{ and } \sum_{\tau_i \in \mathbb{T}} U_i^b \le \rho M_b,\end{aligned} \tag{27}$$

then the greedy type-aware federated scheduling algorithm guarantees to derive a feasible schedule.

Proof. Suppose for contrapositive that the algorithm fails to derive a feasible solution.

As the first case, suppose that this is due to the exclusive allocation phase. By Theorem 13, we know that either $M_a < \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} m_i^a \le \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} \frac{U_i^a}{\rho}$ or $M_b < \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} m_i^b \le \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} \frac{U_i^b}{\rho}$. This concludes the case.

For the second case, the exclusive allocation is successful and, therefore, $M_a^\sharp \ge 0$ and $M_b^\sharp \ge 0$. By Theorem 13,

$$\sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} U_i^a \ge (M_a - M_a^\sharp)\rho \tag{28}$$

$$\sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} U_i^b \ge (M_b - M_b^\sharp)\rho. \tag{29}$$

Suppose that the algorithm fails when it tries to assign task τ_k. There are three further sub-cases:

- Sub-case 1: $\tau_k \in \mathbb{H}^a$, i.e., it cannot find a core q among the $M_b^\sharp$ type b cores to assign τ_k. It holds $\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} U_i^b > \rho M_b^\sharp$ if $M_b^\sharp = 0$ and otherwise by Lemma 15. Together with Equation (29) we obtain

$$\sum_{\tau_i \in \mathbb{T}} U_i^b = \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^b} U_i^b + \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} U_i^b > \rho M_b. \tag{30}$$

- Sub-case 2: $\tau_k \in \mathbb{H}^b$, i.e., it cannot find a core p among the $M_a^\sharp$ type a cores to assign to τ_k. It holds $\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} U_i^a > \rho M_a^\sharp$ if $M_a^\sharp = 0$ and otherwise by Lemma 15. Together with Equation (28) we obtain

$$\sum_{\tau_i \in \mathbb{T}} U_i^a = \sum_{\tau_i \in \mathbb{H}^{ab} \cup \mathbb{H}^a} U_i^a + \sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} U_i^a > \rho M_a. \tag{31}$$

- Sub-case 3: $\tau_k \in \mathbb{LI}$, i.e., it cannot find a pair of cores p, q among the $M_a^\sharp$ type a cores and the $M_b^\sharp$ type b cores to assign to τ_k. If $M_a^\sharp = 0$, then $\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} U_i^a > \rho M_a^\sharp$ and together with Equation (28), we obtain Equation (31). If $M_b^\sharp = 0$, then $\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} U_i^b > \rho M_b^\sharp$ and together with Equation (29), we obtain Equation (30). If $M_a^\sharp > 0$ and $M_b^\sharp > 0$, then we apply Lemma 16. In particular, $\frac{\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^a} U_i^b}{M_b^\sharp} > \rho$ or $\frac{\sum_{\tau_i \in \mathbb{LI} \cup \mathbb{H}^b} U_i^a}{M_a^\sharp} > \rho$ holds and therefore we obtain Equations (31) or (30) as well. Hence, we reach the conclusion that one of the assumptions in Eq. (27) is violated, and the theorem is proved. □

7 IMPROVED SCHEDULING ALGORITHM

The greedy type-aware federated scheduling algorithm in Section 5 is presented based on several rigid "enforcement rules," i.e., choice of parameters, that guide the algorithm to achieve a capacity augmentation bound of 7.25. However, these enforcement rules may lead to poor performance in practical settings [8]. In this section, we introduce an improved type-aware federated scheduling algorithm without the enforcement on the parameters. The improved algorithm determines the execution mode of a task and assigns the task to cores *one task after another* based on four principles. We also prove that the capacity augmentation bound of the improved algorithm remains 7.25.

7.1 Algorithm Description

The improved algorithm works based on four principles:

1) *P-Attempt*: For each task τ_k, the algorithm attempts to determine the execution mode of the task with the following preference order: *Light* > (*Heavy*^{a} ∨ *Heavy*^{b}) > *Heavy*^{ab}. More precisely, when considering task τ_k, the algorithm tries to assign at first τ_k in the *Light* mode, in case of failure, followed by another attempt of the *Heavy*^{a} ∨ *Heavy*^{b} mode. In case executing task τ_k in all of these three modes is not feasible (based on the schedulability tests presented in Section 5.1), task τ_k is assigned to the *Heavy*^{ab} mode, which is validated at the end of the algorithm, i.e., principle *P-Exclusive* here.
2) *P-Share*: The algorithm prefers to *share* cores, i.e., it tries to assign tasks to the shared cores already assigned with certain higher-priority tasks, and only assigns the task to a core without any task assigned on it when such an option is not possible.
3) *P-Efficient*: When a task is in the *Heavy*^{a} or *Heavy*^{b} mode, the number of exclusively allocated cores is minimized just to meet its deadline. That is, the suspension time from core q of type b (symmetrically core p of type a) can be extended as long as the task meets its deadline.

4) *P-Exclusive*: For the tasks that are in the *Heavy*^{ab} mode, there are potentially multiple choices of m_i^a and m_i^b for τ_i. We search for the best combination of them by a combinatorial approach.

Algorithm 1 provides the pseudocode of the improved algorithm. After its initialization, it tries to assign task τ_k by following the *P-Attempt* principle, divided into four blocks: Lines 4 - 8 are for the *Light* mode attempt, Lines 9 - 37 are for the *Heavy*^{a} mode attempt, Lines 38 - 45 are for the *Heavy*^{b} mode attempt, and Line 46 temporarily assigns τ_k to the *Heavy*^{ab} mode. If an earlier attempt is successful, the flag *success* is marked as *true*, and there is no need for subsequent attempts.

Light Mode Attempt $\mathbb{LI}^*$ (Lines 4 - 8). The algorithm first tries to execute task τ_k in the *Light* mode if possible. The function *Light_Attempt* (pseudocode in the Appendix), available in the online supplemental material, returns a pair of cores (p, q) on which τ_k can be feasibly executed on these two cores, by validating the schedulability using Theorem 10. By following the *P-Share* principle, the algorithm prefers sharing cores. That is, whenever possible, task τ_k should be assigned on cores (p, q) which already had certain higher-priority task(s) assigned onto them. Only if this option is not possible, the algorithm tries to assign task τ_k to a core p of type a and/or a core q of type b which was not yet assigned with any higher-priority tasks. If there are multiple valid core pairs, the algorithm applies a fitting strategy to choose one. Recall that finding a pair of cores for a task can be formulated as a bin packing problem, as discussed in Section 5.2. Any fitting strategy can be applied.

Heavy^a Mode Attempt $\mathbb{H}^{a*}$ (Lines 9 - 37). If task τ_k can not be scheduled in the *Light* mode and $C_k^b \leq \rho T_k$, the algorithm tries to schedule the task in the *Heavy*^{a} mode with an objective to minimize the number of exclusively allocated type a cores, following the *P-Efficient* principle. To minimize the number of type a cores exclusively allocated to τ_k, we use a function *HeavyA_Attempt* (pseudocode in the Appendix), available in the online supplemental material, which returns the minimum number of type a cores exclusively needed for task τ_k when its type b execution is assigned on a specified core q, together with the other tasks Ψ_q^b assigned on it. This calculation is based on Lemma 5 and Theorem 9.

It also follows the *P-Share* principle by first trying to find a type b core q^* which already has higher-priority task(s) assigned on it. Therefore, it starts from the *P-Share* principle in Lines 11 - 23 if sharing the execution of task τ_k on a type b core q^* does not result in more than $\left\lceil \frac{C_k^a - L_k^a}{T_k/3 - L_k^a} \right\rceil$ type a cores in the greedy type-aware federated scheduling algorithm. If sharing is not possible, then the algorithm further attempts to assign τ_k on a core q without any higher-priority tasks on it and $\left\lceil \frac{C_k^a - L_k^a}{(T_k - C_k^b) - L_k^a} \right\rceil$ (calculated by the function *HeavyA_Attempt*) exclusively type a cores. If $\rho \leq 1/6$, then the condition $C_k^b \leq \rho T_k$ implies that $\left\lceil \frac{C_k^a - L_k^a}{(T_k - C_k^b) - L_k^a} \right\rceil \leq \left\lceil \frac{C_k^a - L_k^a}{T_k/3 - L_k^a} \right\rceil$.

Heavy^b Mode Attempt $\mathbb{H}^{b*}$. The case of the *Heavy*^{b} mode is symmetric to the case of the *Heavy*^{a} mode. The pseudocode of the counter part can be found in the Appendix, available in the online supplemental material.

Algorithm 1. Improved Type-aware Federated Scheduling

Input: $\mathbb{T}, M_a, M_b, \rho$;
Output: Assignment of tasks to cores
1: $\Psi_p^a \leftarrow \emptyset, \forall p = 1, \ldots, M_a$, $\Psi_q^b \leftarrow \emptyset, \forall q = 1, \ldots, M_b$, $\mathbb{T}' \leftarrow \mathbb{T}$, $\mathbb{H}^{ab*} \leftarrow \emptyset$, $\mathbb{H}^{a*} \leftarrow \emptyset$, $\mathbb{H}^{b*} \leftarrow \emptyset$, $\mathbb{LI}^* \leftarrow \emptyset$; ▷Initialization.
2: **while** $\mathbb{T}'$ is not empty **do**
3: pop out task τ_k with the smallest T_k from $\mathbb{T}'$;
4: **if** the pair $(p, q) \leftarrow$ ***Light_Attempt***(τ_k) can be found **then**
5: $\Psi_p^a \leftarrow \Psi_p^a \cup \{\tau_k\}$ and $\Psi_q^b \leftarrow \Psi_q^b \cup \{\tau_k\}$;
6: $\mathbb{LI}^* \leftarrow \mathbb{LI}^* \cup \{\tau_k\}$;
7: **continue**; ▷*Light* mode successful
8: **end if**
9: **if** $C_k^b \leq \rho T_k$ **then**
10: *success* $\leftarrow$ **false**;
11: **if** Ψ_q^b is $\emptyset$ for every q **then**
12: $m_k^{a*} \leftarrow \infty$;
13: **else**
14: $m_k^{a*} \leftarrow \min_{\Psi_q^b \neq \emptyset}$ ***HeavyA_Attempt***(τ_k, q);
15: $q^* \leftarrow \arg\min_{\Psi_q^b \neq \emptyset}$ ***HeavyA_Attempt***(τ_k, q); ▷ties broken arbitrarily
16: **end if**
17: **if** $m_k^{a*} \leq \lceil (C_k^a - L_k^a)/(T_k/3 - L_k^a) \rceil$ **then** ▷try to assign τ_k with m_k^{a*} type a cores exclusively and to core q^* of type b
18: **if** there are m_k^{a*} cores of p with $\Psi_p^a = \emptyset$ **then**
19: $\Psi_{q^*}^b \leftarrow \Psi_{q^*}^b \cup \{\tau_k\}$;
20: find m_k^{a*} cores of p with $\Psi_p^a = \emptyset$ and set $\Psi_p^a \leftarrow \{\tau_k\}$;
21: *success* $\leftarrow$ **true**;
22: **end if**
23: **end if**
24: **if** *success* is **false** and there exists q with $\Psi_q^b = \emptyset$ **then** ▷try to assign τ_k with m_k^a type a cores exclusively and a new type b core q
25: find a core q with $\Psi_q^b = \emptyset$;
26: $m_k^a \leftarrow$ ***HeavyA_Attempt***(τ_k, q);
27: **if** there are m_k^a cores of p with $\Psi_p^a = \emptyset$ **then**
28: $\Psi_q^b \leftarrow \{\tau_k\}$;
29: find m_k^a cores of p with $\Psi_p^a = \emptyset$, set $\Psi_p^a \leftarrow \{\tau_k\}$;
30: *success* $\leftarrow$ **true**;
31: **end if**
32: **end if**
33: **if** *success* is **true then**
34: $\mathbb{H}^{a*} \leftarrow \mathbb{H}^{a*} \cup \{\tau_k\}$;
35: **continue**; ▷*Heavy*^{a} mode successful
36: **end if**
37: **end if**
38: **if** $C_k^a \leq \rho T_k$ **then**
39: *success* $\leftarrow$ **false**;
40: perform the counterpart of Lines 11 to 32 for the *Heavy*^{b} mode;
41: **if** *success* is **true then**
42: $\mathbb{H}^{b*} \leftarrow \mathbb{H}^{b*} \cup \{\tau_k\}$;
43: **continue**; ▷*Heavy*^{b} mode successful
44: **end if**
45: **end if**
46: $\mathbb{H}^{ab*} \leftarrow \mathbb{H}^{ab*} \cup \{\tau_k\}$; ▷*Heavy*^{ab} mode left for later inspection
47: **end while**
48: $X_a \leftarrow$ the remaining number of type a cores with $\Psi_p^a = \emptyset$;
49: $X_b \leftarrow$ the remaining number of type b cores with $\Psi_p^b = \emptyset$;
50: **if** ***isFeasible_HeavyAB***$(\mathbb{H}^{ab*}, X_a, X_b)$ **then**
51: **return** Ψ_p^a and Ψ_q^b for every p, q;
52: **else**
53: **return** failure;
54: **end if**

Heavy^{ab} Mode (Destiny) $\mathbb{H}^{ab*}$. After examining all the tasks, those that were not assigned to be in the *Light*, *Heavy*^{a}, or *Heavy*^{b} modes have to be checked whether they can be executed in the *Heavy*^{ab} mode, based on the *P-Exclusive* principle. Let X_a and X_b be the remaining number of type a and type b cores with no task assigned on them, respectively. The function *isFeasible_-HeavyAB* builds a dynamic programming table to assign cores to the tasks in $\mathbb{H}^{ab*}$ and validates whether the tasks in $\mathbb{H}^{ab*}$ can be feasibly scheduled under exclusive allocations. Let $P(\mathbb{H},' M_a,' M_b')$ be *True* if the tasks in $\mathbb{H}'$ can be feasibly scheduled under exclusive allocations of M_a' type a cores and M_b' type b cores. As the boundary condition, $P(\emptyset, M_a,' M_b')$ is *True* for any $M_a' \geq 0, M_b' \geq 0$ and $P(\mathbb{H},' M_a,' M_b')$ is *False* if $M_a' < 0$ or $M_b' < 0$. Furthermore, if task τ_k can be feasibly scheduled on m_k^a type a cores and m_k^b type b cores, the function $Sch(\tau_k, m_k^a, m_k^b)$ is *True*; otherwise is *False*.

$$P(\mathbb{H}' \cup \{\tau_k\}, M_a,' M_b') = \bigvee_{m_k^a \geq 0, m_k^b \geq 0} \left\{ P(\mathbb{H},' M_a' - m_k^a, M_b' - m_k^b) \wedge Sch(\tau_k, m_k^a, m_k^b) \right\}.$$

Schedulability test for $Sch(\tau_k, m_k^a, m_k^b)$ can be done by any approaches in Section 4.1. If the adopted test is *self-sustainable* of the schedulability test, we can find feasible assignments of task τ_k by performing binary searches on the number of both core types. Note that there can be multiple pairs of (m_k^a, m_k^b) that $Sch(\tau_k, m_k^a, m_k^b)$ returns *True*, and we only care about the pairs that are not *dominated* by other pairs. We say that a pair (m_k^a, m_k^b) is *dominated* by $(m_k^{a'}, m_k^{b'})$ if $m_k^a \geq m_k^{a'}$ and $m_k^b \geq m_k^{b'}$ since if τ_k is schedulable on $(m_k^{a'}, m_k^{b'})$ cores then it must also be schedulable on (m_k^a, m_k^b) cores due to the *self-sustainable* of the schedulability test.

We apply the standard dynamic programming approach to validate whether $P(\mathbb{H}^{ab*}, X_a, X_b)$ is *True* or not. If it is *True*, we backtrack the dynamic programming table and return a feasible assignment. The proposed algorithm then returns a feasible solution if we can feasibly schedule the tasks in $\mathbb{H}^{ab*}$ and returns failure if not.

Time complexity: The time complexity of the improved type-aware federated scheduling algorithm can be derived as follows. For tasks that share cores, i.e., tasks in the *Light*, *Heavy*^{a}, and *Heavy*^{b} modes, the time complexity is $O(N^2 M_a M_b T_N)$, where T_N is the maximum inter-arrival time of the given N tasks, as discussed in Section 5.2. For tasks with dedicated cores, we can construct the dynamic programming table with time complexity $O(N M_a M_b)$ by keeping only the *non-dominated* entries in the table. The overall time complexity is $O(N^2 M_a M_b T_N)$, which is pseudo-polynomial and can be reduced by approximating the tests in Theorems 9 and 10 in polynomial time [7].

7.2 Capacity Augmentation Bound

We prove that the capacity augmentation bound of the improved type-aware federated scheduling algorithm is still 7.25 in this section. Due to the structural similarity of the greedy type-aware federated scheduling algorithm and the improved version, most of results from Section 6 can be applied with proper restatements.

Lemma 18. (Extension of Lemma 12 *) For any $0 < \rho \leq 1/6$, when $0 < L_i^a \leq \rho T_i < C_i^a$ and $0 < L_i^b \leq \rho T_i < C_i^b$, let $m_i^{a'}$ (respectively, $m_i^{b'}$) be the number of type a (respectively type b) cores allocated to τ_i if task τ_i is successfully assigned to $\mathbb{H}^{a*}$ (respectively $\mathbb{H}^{b*}$) in Algorithm 1, we have*

$$m_i^{a'} \leq \frac{U_i^a}{\rho}, \forall \tau_i \in \mathbb{H}^{a*} \quad \text{and} \quad m_i^{b'} \leq \frac{U_i^b}{\rho}, \forall \tau_i \in \mathbb{H}^{b*}. \tag{32}$$

Proof. As Algorithm 1 does not allocate more than $m_i^a = \left\lceil \frac{U_i^a - \frac{L_i^a}{T_i}}{\frac{1}{3} - \frac{L_i^a}{T_i}} \right\rceil$ cores for τ_i when $\tau_i \in \mathbb{H}^{a*}$, with the same procedure of the proof of Lemma 12, we reach the conclusion. □

Lemma 19. **(Extension of Lemma 16***) For any $0 < \rho \leq 1/7.25$, let task τ_k be first light task in Algorithm 1, which is classified as a light task in $\mathbb{LI}$ but cannot be feasibly assigned during the* Light mode attempt, *i.e., not in $\mathbb{LI}^*$. If such a task τ_k exists, then*

$$\frac{\sum_{\tau_i \in \mathbb{LI}^* \cup \mathbb{H}^{a*}} U_i^b}{M_b} + \frac{\sum_{\tau_i \in \mathbb{LI}^* \cup \mathbb{H}^{b*}} U_i^a}{M_a} > 2\rho. \tag{33}$$

Proof. The same proof procedure of Lemma 16 can be applied by replacing $M_a^\sharp$ with M_a and $M_b^\sharp$ with M_b and by adopting Lemma 18 instead of Lemma 12. □

Lemma 20. **(Extension of Lemma 15)** *For any $0 < \rho \leq 1/6$, when $0 < L_i^a \leq \rho T_i$ and $0 < L_i^b \leq \rho T_i$ for every task $\tau_i \in \mathbb{T}$, if task τ_k in $\mathbb{H}^a$ is the first task that cannot not be feasibly scheduled after the Heavy^a mode attempt under Algorithm 1, then*

$$U_k^b + \sum_{\tau_i \in \mathbb{LI}^* \cup \mathbb{H}^{a*} \cup \mathbb{H}^{b*}} U_i^b > \rho M_b. \tag{34}$$

or

$$U_k^a + \sum_{\tau_i \in \mathbb{LI}^* \cup \mathbb{H}^{a*} \cup \mathbb{H}^{b*}} U_i^a > \rho M_a. \tag{35}$$

The same condition holds, if task τ_k in $\mathbb{H}^b$ is the first task that cannot not be feasibly scheduled after the Heavy^b mode attempt under Algorithm 1.

Proof. We prove the case that task τ_k in $\mathbb{H}^a$, as the other case is symmetric. In this case, $U_k^a > \rho$. The reason why task τ_k is not assigned to core q is because the procedure between Line 9 and Line 32 in Algorithm 1 fails. There are two scenarios of such a failure:

- *Insufficient* type a cores are available: As Algorithm 1 attempts to use up to $m_k^a = \left\lceil \frac{C_k^a - L_k^a}{\frac{T_k}{3} - L_k^a} \right\rceil$ type a cores. In this case, $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^a > (M_a - m_k^a)\rho$, leading to Eq. (35).
- Task τ_k cannot be assigned to any type b core q under the schedulability test: In this case, the same procedure in the proof of Lemma 15 can be applied, leading to $\sum_{\tau_i \in \mathbb{LI}^* \cup \mathbb{H}^{a*} \cup \mathbb{H}^{b*}} U_i^b > \rho M_b$, as well as Eq. (34). □

Lemma 21. *Let X_a and X_b be the integers defined in Line 48 and Line 49 of Algorithm 1, respectively. When ρ is set to $1/7.25$*

and $\max\{L_i^a, L_i^b\} \le L_i \le \rho T_i$*, after Line 49 of Algorithm 1, one of the following three conditions holds:*

1) $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^a > \rho M_a$,
2) $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^b > \rho M_b$, *or*
3) $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^a > \rho(M_a - X_a - 1)$ *and* $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^b > \rho(M_b - X_b - 1)$.

Proof. Suppose that right after assigning task τ_k there are two type b cores with core utilization > 0 and $< \rho$. In Appendix, available in the online supplemental material, we show that this is not possible unless $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^a > \rho M_a$. The condition that there is at most one type b core with core utilization > 0 and $< \rho$ implies that $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^b > \rho(M_b - X_b - 1)$ holds.

The symmetric part of $\sum_{\tau_i \in \mathbb{H}^{a*} \cup \mathbb{H}^{b*} \cup \mathbb{LI}^*} U_i^a > \rho(M_a - X_a - 1)$ holds with the same proof procedure. Therefore, one of the three conditions in the statement of the lemma holds. □

Theorem 22. *The capacity augmentation bound of the improved type-aware Federated Scheduling algorithm is at most 7.25 when* ρ *is set to* $1/7.25$.

Proof. The proof is similar to that of Theorem 17 by contrapositive. Suppose that Algorithm 1 fails to assign a certain task in $\mathbb{T}$.

- *Case 1*: $\mathbb{LI} \setminus \mathbb{LI}^* \neq \emptyset$, i.e., a light task τ_k fails to be assigned after the *Light* mode attempt: In this case, Eq. (33) from Lemma 19 concludes the capacity augmentation bound.
- *Case 2*: $\mathbb{H}^a \setminus (\mathbb{LI}^* \cup \mathbb{H}^{a*}) \neq \emptyset$, i.e., a *Heavy*^{a} task τ_k fails to be assigned after the *Light* mode and the *Heavy*^{a} mode attempts: In this case, Lemma 20 concludes the capacity augmentation bound.
- *Case 3*: $\mathbb{H}^b \setminus (\mathbb{LI}^* \cup \mathbb{H}^{b*}) \neq \emptyset$, i.e., a *Heavy*^{b} task τ_k fails to be assigned after the *Light* mode and the *Heavy*^{b} mode attempts: In this symmetric case, Lemma 20 concludes the capacity augmentation bound.
- *Case 4*: $(\mathbb{LI} \cup \mathbb{H}^a \cup \mathbb{H}^b) \setminus (\mathbb{LI}^* \cup \mathbb{H}^{a*} \cup \mathbb{H}^{b*}) = \emptyset$, i.e., all *Light* tasks, *Heavy*^{a} tasks, and *Heavy*^{b} tasks are successfully assigned to cores by Line 48 of Algorithm 1. In this case, $\mathbb{H}^{ab*} \subseteq \mathbb{H}^{ab}$ and every task τ_i in $\mathbb{H}^{ab*}$ that has $U_i^a > \rho$ and $U_i^b > \rho$. The improved algorithm fails to derive a solution when $P(\mathbb{H}^{ab*}, X_a, X_b)$ is False due to insufficient amount of cores. One particular solution is to assign $m_i^a = \left\lceil \frac{C_i^a - L_i^a}{\frac{T_i}{2} - L_i^a} \right\rceil$ type a cores and $m_i^b = \left\lceil \frac{C_i^b - L_i^b}{\frac{T_i}{2} - L_i^b} \right\rceil$ type b cores for every task τ_i in $\mathbb{H}^{ab*}$. In this case, $P(\mathbb{H}^{ab*}, \sum_{\tau_i \in \mathbb{H}^{ab*}} m_i^a, \sum_{\tau_i \in \mathbb{H}^{ab*}} m_i^b)$ is guaranteed return True. Therefore, since m_i^a and m_i^b are integers, either $X_a < X_a + 1 \le \sum_{\tau_i \in \mathbb{H}^{ab*}} m_i^a$ or $X_b < X_b + 1 \le \sum_{\tau_i \in \mathbb{H}^{ab*}} m_i^b$

  By applying Lemma 11 we further know that either $\sum_{\tau_i \in \mathbb{H}^{ab*}} \frac{U_i^a}{\rho} \ge \sum_{\tau_i \in \mathbb{H}^{ab*}} m_i^a \ge X_a + 1$ or $\sum_{\tau_i \in \mathbb{H}^{ab*}} \frac{U_i^b}{\rho} \ge \sum_{\tau_i \in \mathbb{H}^{ab*}} m_i^b \ge X_b + 1$. Together with the three conditions in Lemma 21, we conclude that either $\sum_{\tau_i \in \mathbb{T}} U_i^a > \rho M_a$ or $\sum_{\tau_i \in \mathbb{T}} U_i^b > \rho M_b$.

Due to the above cases, the capacity augmentation bound of 7.25 is proven as the contradiction is reached. □

8 EVALUATION

In this section, we conduct an experimental evaluation of our proposed algorithms on synthetic task sets.

8.1 Environment Setting

Given M_a type a cores, M_b type b cores, and the target utilization U, we generate a task set as follows. The number of DAG tasks N is selected uniformly at random from $[\frac{1}{2} \times \max(M_a, M_b),\ 2 \times \max(M_a, M_b)]$. We apply the Dirichlet-Rescale (DRS) algorithm [12] to determine the utilization for each of the N tasks. The period of a task, i.e., T_i, is selected uniformly at random from $[100, 1000]$. To generate a DAG, we first determine the number of nodes by selecting a number uniformly at random from $[\frac{1}{2} \times (M_a + M_b),\ 5 \times \max(M_a, M_b)]$, and apply DRS to generate the utilization for each node. The $G(n, p)$ algorithm [10] is used to generate the edges between nodes, with probability $p_e \in [0.1, 0.9]$. We generate task sets with different target utilization, from 0 to $60\% \times (M_a + M_b)$ with step of $5\% \times (M_a + M_b)$. For each target, 100 sets are generated and stored as `pure_task_sets`. The type of each node in the `pure_task_sets` is determined as follows. To evaluate the effect of the *skewness* of the workload, we introduce two parameters: r controls the share of tasks that have a skewed workload; and P_ℓ controls the skewness of the skewed tasks. More specifically, in each task set, we pick $r\%$ of the tasks to be *skewed tasks*. Skewed tasks are determined to be type a skewed with a probability of 50% and type b skewed otherwise. For a type a skewed task, $P_\ell\%$ of the nodes in the task are selected uniformly at random to be in type b, while the rest of the nodes are assigned to type a. For non-skewed tasks, each node in a task has a probability $M_a/(M_a + M_b)$ of being assigned type a and a probability $M_b/(M_a + M_b)$ of type b. Note that assigning types to nodes does not change the structure of a DAG. We use task sets with different r and P_ℓ as inputs in our evaluation. Note that this setup may result in many skewed tasks (depending on parameter r), but the workload of a task set as a whole is still expected to be balanced.

We apply the following algorithms in our evaluation:

- *FED-IMPROVED*: The improved type-aware federated scheduling algorithm proposed in Section 7 with Theorems 8 and 9 as schedulability tests for tasks in the *Heavy*^{ab} and *Heavy*^{a}/*Heavy*^{b} modes, respectively.
- *FED-GREEDY*: The greedy type-aware federated scheduling algorithm proposed in Section 5.2.
- *HAN-EMU/GREEDY*: federated scheduling algorithms proposed by Han et al. in [14]. *HAN-EMU* enumerates all possible combinations of (m_a, m_b) for each *heavy* task, while *HAN-GREEDY* applies a penalty-based greedy algorithm to find a feasible assignment. Note that their schedulability analysis for *light* tasks in Lemma 5.1 in [14] is unsafe as they only considered the self-suspending behavior of the task under analysis but *not* the higher-priority self-suspending tasks. In addition, their analysis did *not* consider carry-in jobs, as

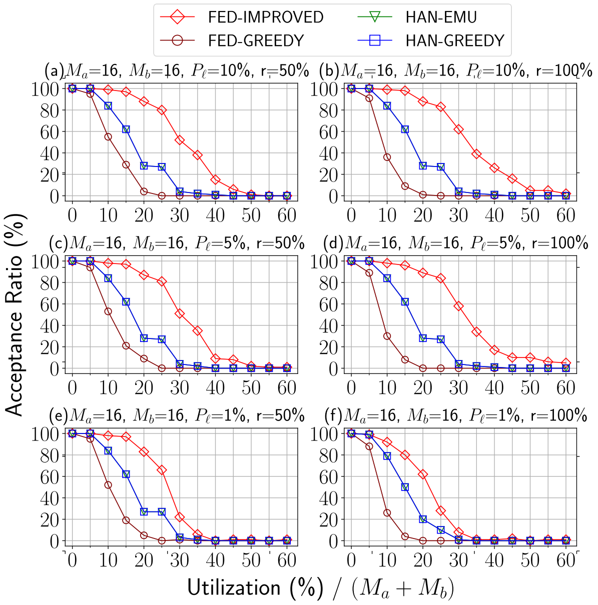


Fig. 3. Schedulability of different approaches when $M_a = M_b$.

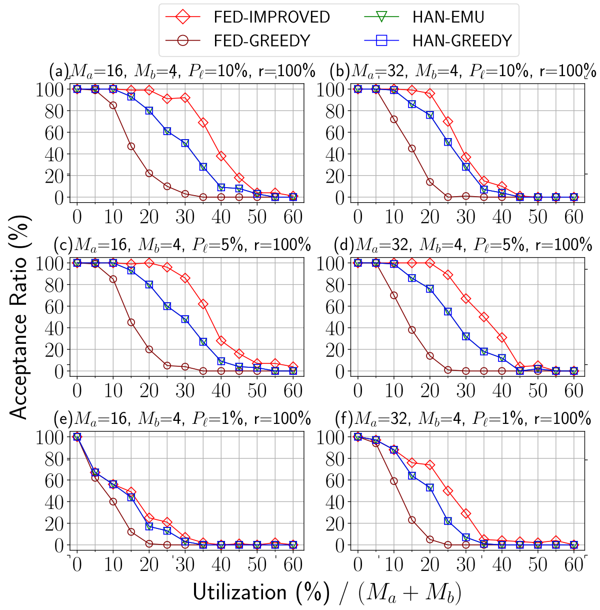


Fig. 4. Schedulability of different approaches when $M_a \neq M_b$.

described by Baker [1]. In our evaluation, the *light* tasks are scheduled based on partitioned scheduling as also presented in our paper, for fairness. Specifically, Sun and Di Natale [22] demonstrated that the global rate-monotonic scheduling strategy can be converted to a partitioned rate-monotonic schedule without sacrificing the schedulability.

8.2 Schedulability

We conducted experiments for $M_a, M_b \in \{4, 16, 32\}$. We investigate different values for the parameter $r \in \{0\%, 50\%, 100\%\}$ and $P_\ell \in \{10\%, 5\%, 1\%\}$ in our evaluation. When $M_a = M_b = 16$, Fig. 3 shows that *FED-IMPROVED* outperforms the two methods proposed by Han et al. significantly in all the evaluated cases. When the share of *skewed* tasks increases, i.e., r increases from 50% to 100% (Figs. 3a $\rightarrow$ 3b and 3c $\rightarrow$ 3d) the performance of none of the evaluated methods is affected significantly. However, when P_ℓ is decreased to an extremely small value, i.e., $P_\ell = 1\%$, (Figs. 3e and 3f), all the evaluated methods suffer from performance degradation.

Next, we compare the schedulability of the algorithms in skewed systems for $M_a \in \{16, 32\}$ and $M_b = 4$. Fig. 4 shows that *FED-IMPROVED* outperforms other methods significantly in most of the evaluated cases. However, in Fig. 4e, none of the methods work well due to the extremely unbalanced type configuration, i.e., $P_\ell = 1\%$ and $r = 100\%$. When P_ℓ is slightly smaller than $M_b/(M_a + M_b)$ (Figs. 4a, 4c, and 4d), *FED-IMPROVED* dominates other methods due the ability to handle the $heavy^a$ tasks. Although *FED-GREEDY* has a constant capacity augmentation bound, the decisions about the execution mode and the number of exclusively allocated cores are purely based on the parameters of a task, which *"may come at the expense of poor performance in practical settings"* as pointed out in [8], c.f. Observation 6 in [8]. The two methods proposed by Han et al. have the same performance in both experiments since the greedy algorithm *HAN-GREEDY* finds the same optimal combination for the *heavy* tasks as the enumeration algorithm *HAN-EMU*.

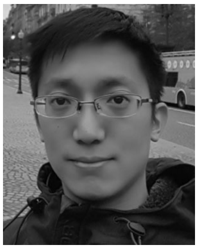
9 Conclusion

In this paper, we introduce type-aware federated scheduling algorithms for sporadic typed DAG tasks with implicit deadlines on heterogeneous multicore platforms with two types of cores. Each task is executed in one of four modes: *Heavy*^{ab}, *Heavy*^{a}, *Heavy*^{b}, and *Light*. Our proposed algorithm determines the execution mode for each task, and schedules tasks in each mode accordingly. We prove that the capacity augmentation bound of the proposed greedy algorithm is 7.25. Furthermore, we improve the algorithm by eliminating enforcement rules. The improved algorithm maintains a 7.25 capacity argumentation bound, but is shown to exhibit better performance in our experimental evaluation.

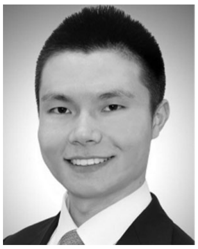
References

[1] T. P. Baker, "Multiprocessor EDF and deadline monotonic schedulability analysis," in *Proc. IEEE Real-Time Syst. Symp.*, 2003, pp. 120–129.
[2] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *Proc. IEEE 27th Int. Real-Time Syst. Symp.*, 2006, pp. 159–168.
[3] S. Baskiyar and R. Abdel-Kader, "Energy aware DAG scheduling on heterogeneous systems," *Clust. Comput.*, vol. 13, no. 4, pp. 373–383, 2010.
[4] J.-J. Chen, "Federated scheduling admits no constant speedup factors for constrained-deadline dag task systems," *Real-Time Syst.*, vol. 52, no. 6, pp. 833–838, Nov. 2016.
[5] J.-J. Chen, W.-H. Huang, and C. Liu, "k2U: A general framework from k-point effective schedulability analysis to utilization-based tests," in *Proc. Real-Time Syst. Symp.*, 2015, pp. 107–118.
[6] J.-J. Chen, W.-H. Huang, and C. Liu, "Automatic parameter derivations in k2U framework," 2016, *arXiv:1605.00119*.
[7] J.-J. Chen, G. Nelissen, and W.-H. Huang, "A unifying response time analysis framework for dynamic self-suspending tasks," in *Proc. Euromicro Conf. Real-Time Syst.*, 2016, pp. 61–71.
[8] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and R. I. Davis, "On the pitfalls of resource augmentation factors and utilization bounds in real-time scheduling," in *Proc. Euromicro Conf. Real-Time Syst.*, 2017, pp. 9:1–9:25.
[9] K. Chronaki et al., "Task scheduling techniques for asymmetric multi-core systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2074–2087, Jul. 2017.
[10] P. Erdös, "On random graphs I," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.

[11] R. L. Graham, “Bounds on multiprocessing timing anomalies,” *SIAM J. Appl. Math.*, vol. 17, no. 2, pp. 416–429, 1969.
[12] D. Griffin, I. Bate, and R. I. Davis, “Generating utilization vectors for the systematic evaluation of schedulability tests,” in *Proc. IEEE 41st Real-Time Syst. Symp.*, 2020, pp. 76–88.
[13] M. Han, N. Guan, J. Sun, Q. He, Q. Deng, and W. Liu, “Response time bounds for typed DAG parallel tasks on heterogeneous multi-cores,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2567–2581, Nov. 2019.
[14] M. Han, T. Zhang, Y. Lin, and Q. Deng, “Federated scheduling for typed DAG tasks scheduling analysis on heterogeneous multi-cores,” *J. Syst. Archit.*, vol. 112, 2021, Art. no. 101870.
[15] W.-H. Huang, J.-J. Chen, and J. Reineke, “MIRROR: Symmetric timing analysis for real-time tasks on multicore platforms with shared resources,” in *Proc. Des. Automat. Conf.*, 2016, pp. 158:1–158:6.
[16] D. Kim, Y. Ko, and S. Lim, “Energy-efficient real-time multi-core assignment scheme for asymmetric multi-core mobile devices,” *IEEE Access*, vol. 8, pp. 117324–117334, 2020.
[17] J. Li, K. Agrawal, C. Lu, and C. D. Gill, “Analysis of global EDF for parallel tasks,” in *Proc. Euromicro Conf. Real-Time Syst.*, 2013, pp. 3–13.
[18] S. Moulik, R. Devaraj, and A. Sarkar, “HEALERS: A heterogeneous energy-aware low-overhead real-time scheduler,” *IET Comput. Digit. Tech.*, vol. 13, no. 6, pp. 470–480, 2019.
[19] NVIDIA. NVIDIA Tegra K1 SoC. Accessed: Sep. 7, 2022. [Online]. Available: https://developer.nvidia.com/embedded/tegra-k1
[20] F. Rossi, P. van Beek, and T. Walsh, Ed. *Handbook of Constraint Programming, vol. 2 of Foundations of Artificial Intelligence*. Amsterdam, Netherlands: Elsevier, 2006.
[21] Samsung. Samsung Exynos series. Accessed: Sep. 7, 2022. [Online]. Available: https://semiconductor.samsung.com/processor/
[22] Y. Sun and M. D. Natale, “Assessing the pessimism of current multicore global fixed-priority schedulability analysis,” in *Proc. ACM Symp. Appl. Comput.*, 2018, pp. 575–583.
[23] V. G. Timkovsky, “Is a unit-time job shop not easier than identical parallel machines?,” *Discrete Appl. Math.*, vol. 85, no. 2, pp. 149–162, 1998.
[24] Xilinx. Xilinx versal series. Accessed: Sep. 7, 2022. [Online]. Available: https://www.xilinx.com/products/silicon-devices/acap/versal.html



Ching-Chi Lin (Member, IEEE) received the MSc and PhD degrees in computer science and information engineering from National Taiwan University, Taiwan, in 2010 and 2018, respectively. He is a postdoc with the chair for Design Automation of Embedded Systems (DAES), TU Dortmund University, Germany, since 2020. His research interests include parallel computing, scheduling algorithms, and distributed systems.



Junjie Shi (Student Member, IEEE) received the master’s degree in electronic technology and information technology from TU Dortmund University, Germany, in 2017, and currently working toward the PhD degree with TU Dortmund University. His research interests are resource-sharing protocols for real-time systems, resource aware scheduling for machine learning algorithms, and computation offloading for real-time systems.

Niklas Ueter received the master’s degree in computer science from TU Dortmund University, Germany, in 2018 and now is working toward the PhD degree with TU Dortmund University, supervised by Prof. Dr. Jian-Jia Chen. His research interests are in 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, the University of Duisburg-Essen, Germany, in 2019. Since that time he 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 is in the area of embedded and real-time systems. Currently, he focuses his research on the schedulability analysis of self-suspending task sets.

Jan Reineke received the BSc degree in computing science from the University of Oldenburg, in 2003, and the MSc and PhD degrees in computer science with Saarland University, in 2005 and 2008, respectively. He is a Professor of computer science with Saarland University, Germany. Before joining Saarland University, in 2012, he was a postdoctoral scholar with UC Berkeley from 2009 to 2011. His research centers around problems with the boundary between hardware and software. In 2012, he was selected as an Intel Early Career Faculty Honor Program Awardee. He was the PC chair of EMSOFT 2014, the International Conference on Embedded Software, a Topic co-chair with DATE 2016 and the PC chair of WCET 2017, the International Workshop on Worst-Case Execution Time Analysis. His papers have been Awarded 7 paper awards, most recently at Oakland (2021), RTSS (2018, 2019), and ECRTS (2017). In 2021, he has been Awarded an ERC Advanced Grant.

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, Germany. He was Junior Professor with the Department of Informatics, Karlsruhe Institute of Technology (KIT), 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.