Reservation-Based Federated Scheduling for Parallel Real-Time Tasks

Niklas Ueter\textsuperscript{1}, Georg von der Bruggen\textsuperscript{1}, Jian-Jia Chen\textsuperscript{1}, Jing Li\textsuperscript{2}, and Kunal Agrawal\textsuperscript{3}

\textsuperscript{1} TU Dortmund University, Germany
\textsuperscript{2} New Jersey Institute of Technology, USA
\textsuperscript{3} Washington University in St. Louis, USA

Abstract—Multicore systems are increasingly utilized in real-time systems in order to address the high computational demands. To fully exploit the advantages of multicore processing, possible intra-task parallelism modeled as a directed acyclic graph (DAG) must be utilized efficiently. This paper considers the scheduling problem for parallel real-time tasks with constrained and arbitrary deadlines. In contrast to prior work in this area, it generalizes federated scheduling and proposes a novel reservation-based approach. Namely, we propose a reservation-based federated scheduling strategy that reduces the problem of scheduling arbitrary-deadline DAG task sets to the problem of scheduling arbitrary-deadline sequential task sets by allocating reservation servers. We provide the general reservation design for sporadic parallel tasks, such that any scheduling algorithm and analysis for sequential tasks with arbitrary deadlines can be used to execute the allocated reservation servers of parallel tasks. Moreover, the proposed reservation-based federated scheduling algorithms provide constant speedup factors with respect to any optimal scheduler for arbitrary-deadline DAG task sets. We demonstrate via numerical and empirical experiments that our algorithms are competitive with the state of the art.

Index Terms—Parallel Real-Time Tasks, DAG, Federated Scheduling, Partitioned Scheduling, Servers

I. INTRODUCTION

Modern real-time systems increasingly facilitate multicore systems to account for the growing computational demands of real-time applications. In uniprocessor platforms, the execution-demand of sequential real-time tasks is solely modeled by their worst-case execution times, since the processor executes only one job at each point in time and thus there is no need to express potential parallel execution paths. In contrast, multicore platforms allow \textit{inter-task parallelism}, i.e., to execute sequential programs concurrently, and \textit{intra-task parallelism}, i.e., to execute a job of a parallelized task on multiple processors (cores) at the same time. To enable intra-task parallelism, potentially parallel execution of an application must be considered at design time. As demonstrated by Serrano et al. [38], the asynchronous parallel task model, represented as a directed acyclic graph (DAG) where each thread corresponds to a node and the edges denote precedence constraints, can be realized using the untied task model of OpenMP. Due to this fact and the increasing OpenMP support by newly developed multicore processors [38], the DAG task model is a reasonable parallel real-time task model to be used for the design of scheduling algorithms.

A recent approach for scheduling parallel real-time DAG tasks is federated scheduling by Li et al. [31], where heavy tasks, i.e., tasks that need to execute on more than one processor simultaneously in order to meet their relative deadlines, are assigned a set of processors exclusively, whilst the light tasks are sequentialized and scheduled on the remaining processors. The exclusive assignment allows a simple response-time analysis due to the absence of inter-task interference and contiguous service provided to the DAG task by the assigned processors. Additionally, limiting the number of processors that a DAG task can execute on results in lower synchronization and scheduling overheads, and the non-preemption of the heavy tasks has potential cache advantages. On the other hand, the exclusive granting of processors to a heavy task can potentially waste many system resources when a job of the task completes before the release of the next one, especially for constrained- and arbitrary-deadline tasks (an example is provided in [16]).

To address these limitations of federated scheduling, whilst maintaining the favorable analytical properties of absent inter-task interference, we provide the following results:

- We propose a novel \textit{reservation-based federated scheduling} approach for scheduling sporadic DAG task sets with arbitrary deadlines in Section IV. That is, each DAG task is assigned a set of dedicated reservation servers that inherit the timing models of the associated DAG task. These reservation servers can then be scheduled as sequential tasks by any multiprocessor scheduling algorithm that supports such timing models.

- We provide constraints and design rules for the dimensioning and assignment of the reservation servers, such that each DAG task is schedulable if all its reservation servers are schedulable under any multiprocessor scheduling algorithm that has a corresponding analysis guaranteeing the schedulability of the aforementioned servers. Specifically, in Section V we present provably sufficient reservations and different reservation assignment algorithms. One of them, to the best of our knowledge, is the first algorithm for scheduling DAG task sets with arbitrary deadlines that has a constant speedup factor.

- Additionally, we resolve the problem of non-constant speedup factors for arbitrary-deadline DAG task sets under federated scheduling with respect to any optimal DAG task set scheduling algorithm as pointed out by Chen [16]. We show that the speedup factor of our approach is at most \(3 + 2\sqrt{2}\) by the design of a specific set of reservation servers that are scheduled under partitioned
and global multiprocessor scheduling in Sections VI and VII, respectively. We further improve the reservation assignment algorithm in Section VIII. Note that our treatment for arbitrary-deadline task systems may result in different jobs from a task being executed at the same time since our reservation-based strategies provide the reservation immediately when a DAG job arrives.

- Finally, we conduct various numerical and empirical experiments in Section IX and show that our algorithms are comparable to the state of the art for implicit-deadline tasks. In addition, our algorithms have the capability of handling arbitrary-deadline tasks and scheduling constrained-deadlines tasks with stringent timing requirement efficiently.

II. RELATED WORK

In this section, we review the prior work that is most related to this research. The problem of scheduling parallel-real-time DAG tasks has been broadly researched. To the best of our knowledge, three types of scheduling approaches exist.

Decomposition-based strategies: A DAG task is decomposed into a set of sequential subtasks with specified relative deadlines and offsets of their release times, where the internal dependencies are maintained by the decomposition. These sequential subtasks are then scheduled accordingly without considering the DAG structure anymore, e.g., [27]–[29], [36], [37]. Decomposition-based strategies must utilize the DAG structure off-line in order to apply the decomposition.

Scheduling without any treatment: None of the parameters of a task nor its DAG structure is used at all when making the scheduling decisions. The subtasks of a job have the same static or dynamic priority as the job. Whenever a subtask is ready to be executed, the standard global or partitioned multiprocessor scheduling is used to schedule the subtasks, e.g., [2], [14], [30], [31]. Moreover, when the DAG task is further modelled with conditional branches, global scheduling has been also studied in [6], [34], [35].

Federated scheduling: The task set is partitioned into light and heavy tasks based on their utilizations. Light tasks are those that can be completely sequentialized and still meet their deadlines on one processor. Heavy tasks are those that need more than one processor to meet their deadlines. In the original design of federated scheduling for implicit-deadline task systems proposed by Li et al. [31], a light task is solely executed sequentially without exploiting the parallelized structure, and a heavy task is assigned to its designated processors that exclusively execute only this heavy task. Jiang et al. [26] extended this approach to semi-federated scheduling, in which one or two processors, used by a heavy task, can be shared with other tasks. Baruah [3]–[5] adopted the concept of federated scheduling for scheduling constrained- and arbitrary-deadline task systems. However, Chen [16] later showed that federated scheduling does not have any constant speedup factor with respect to the optimal scheduling algorithm for constrained- and arbitrary-deadline tasks. As discussed earlier, despite the advantages of federated and semi-federated scheduling, it cannot fully utilize the processors when tasks have constrained or arbitrary deadlines due to the dedicated processors assigned to heavy tasks. Chen in [16] provided a concrete task set to further explain this issue.

A related but different problem that has been extensively studied is the hierarchical scheduling for real-time tasks in virtual machines on multiprocessors [12], [15], [33], [39], [41]. Most of these works model this problem using the compositional analysis framework where virtual processors of a virtual machine are abstracted using interfaces with certain guaranteed capacity, and the schedulability of tasks running within each component can be analyzed independently. Although some of these interfaces are also named as resource reservations or servers, their focus is on first designing an appropriate interface that can accurately describe the virtual resources yet is simple to use and then providing schedulability tests to tasks over that interface. In addition, none of these works considers parallel real-time tasks.

III. PARALLEL REAL-TIME TASK MODEL

This work considers the real-time scheduling problem on a multiprocessor system comprised of M homogeneous (identical) processors. We adopt the widely used sporadic task model to describe the real-time tasks, where a task $\tau_i$ is characterized by its relative deadline $D_i$, its period (minimum inter-arrival time) $T_i$, and its worst-case execution time (WCET) $C_i$. A sporadic task is an infinite sequence of task instances, referred to as jobs, where the arrival of two consecutive jobs of a task is separated at least by its minimum inter-arrival time, i.e., the arrival rate is limited. To fulfill its timing requirement, a job of task $\tau_i$ must finish at most $C_i$ units of computation between the arrival of a job at $t_a$ and that jobs absolute deadline at $t_a + D_i$. A sporadic task system $T$ is called an implicit-deadline system if $D_i = T_i$ holds for each $\tau_i$ in $T$ and is called a constrained-deadline system if $D_i \leq T_i$ holds for each $\tau_i$ in $T$. Otherwise, $T$ is an arbitrary-deadline task system.

Through out this paper, we focus on how to schedule parallel real-time tasks that have internal parallelism and allow subtasks to be executed simultaneously on multiple processors. An established model for parallel tasks is the Directed-Acyclic-Graph (DAG) model, where the execution of task $\tau_i$ is divided into subtasks and the precedence constraints of these subtasks are defined by a DAG structure. Each node of a DAG represents a subtask that can be executed whenever all precedence constraints are met, i.e., all directly incident jobs finished their executions.

Two parameters are used to characterize a DAG task $\tau_i$:

- Total execution time (or work) $C_i$: the summation of the worst-case execution times of all the subtasks of task $\tau_i$.
- Critical-path length $L_i$: the length of the critical path in the given DAG, i.e., the worst case execution time of the task on an infinite number of processors.

By definition, $C_i \geq L_i > 0$ for every task $\tau_i$. Furthermore, the utilization of $\tau_i$ is denoted by $U_i = \frac{C_i}{T_i}$.

The abstraction of using work and critical-path length to describe the DAG has the advantage that it is completely agnostic of the internal parallelization structure, i.e., how many
subtasks exist and how the precedence constraints amongst
them are. Scheduling algorithms that can feasibly schedule
DAG task sets solely based on these two parameters also allow
a task to change the DAG structure during runtime as long
as those parameters persist. The apparent downside of this
abstraction is the pessimism since the worst possible structure
has to be considered regardless of the actual structure, i.e., the
scheduling algorithms have to suffice the tasks deadline for all
possible structures under given parameter constraints.

We assume that the worst-case execution time already covers
the worst-case combination of the interference in the memory
and bus contention, similar to all the approaches in this research
time to schedule DAG tasks, e.g., [26], [31], [35]. We note that
such a safe bound may introduce pessimism into the analysis.

IV. RESERVATION-BASED FEDERATED SCHEDULING

In this paper, we propose to use reservation-based allocation
instead of exclusive allocation for parallel tasks with con-
strained or arbitrary deadlines. That is, instead of dedicating a
few processors to a heavy task, we assign a few reservation
servers to a heavy task. The timing properties of a DAG task
will be guaranteed as long as the corresponding reservations
can be guaranteed to be feasibly provided. We now describe
the basic ideas behind reservation-based federated scheduling
for DAG tasks and some constraints on the reservation design.

A. Reservation Server Design

The reservation-based federated scheduling is a natural
generalization of the federated scheduling approach, where
sufficient exclusive processor time budgets (but not necessarily
entire processors) are reserved for heavy tasks to execute
potentially in parallel and to meet their deadlines. Therefore,
it is a hierarchical scheduling strategy: scheduling a DAG task
inside the reserved resources, namely reservation servers, and
scheduling the reservation servers on the multiprocessors.

In Federated Scheduling, heavy tasks are allocated to proces-
sors exclusively based on the DAG task’s density, which may
result in very low utilization of the processors in constrained-
deadline task systems due to DAG tasks with high density and
low utilization, i.e., the density \( C_i / D_i \) is large compared to the
utilization \( C_i / T_i \). Thus, the provided 100% utilization of
each allocated processor cannot be benefited from.

In consequence, speeding up processors and thus decreasing
the DAG task’s execution-time does not improve schedulability
if the system can not provide a sufficient number of processors.
This issue is further explained in [16]. Contrary, in our
reservation-based approach, we relate the number of in-parallel
required service and minimal-sequential service, and are thus
able to decouple reservation-server utilization (with a constant
inflation factor \( \gamma \)) in trade for more in-parallel service. By
enforcing this constant inflation factor for each heavy task, we
can relate the reservation demands to constant speedup factors.

In reservation-based scheduling, each DAG task \( \tau_i \) gets
\( m_i \) sequential reservation servers with budgets (service provi-
sioning) \( E_{i,1}, \ldots, E_{i,t}, \ldots, E_{i,m_i} \). We enforce that the reservation
servers are synchronous with the release of a DAG task’s job
and the reservation servers corresponding to a DAG task \( \tau_i \)
have the same relative deadline and inter-arrival time as \( \tau_i \).
This means that the release pattern of a reservation server is
inherited from the DAG task and whenever a DAG task releases
a job at \( t_r \), the associated service is provided in the release-
and deadline-interval \( [t_r, t_r + D_i] \).

We now formally define the reservation server as follows:

**Definition 1:** The \( \ell \)-th reservation server \( \tau_{i,\ell} \) for serving a
DAG task \( \tau_i \) is defined by the tuple \( (E_{i,\ell}, D_i, T_i) \), such that
\( E_{i,\ell} \) amount of service (computation time) is provided to the
DAG task \( \tau_i \) over the interval \( [t_r, t_r + D_i] \) with a minimum
inter-arrival time of \( T_i \).

In order to analytically treat a reservation server as if it is
a sporadic sequential task, we enforce the reservation to be
active, i.e., eligible for scheduling, until the whole runtime
budget is depleted. This also allows us to schedule multiple
reservation servers in parallel instead of sequencing multiple
pending servers of the same tasks in a first-in-first-out manner.
However, the reservation servers that are released at time \( t_r \)
for a given DAG task \( \tau_i \) used to serve the DAG job of task \( \tau_i \) that
arrived at time $t_r$ exclusively. This implies that multiple consecutive
DAG jobs may be executing in parallel. Note that we do not
restrict any reservation server to service certain subtasks of
a task (nodes in the DAG) exclusively. Instead, reservation
servers of the same task are eligible to service any subtask as
long as they belong to the job that initiated the activation of
the reservations. Each DAG job is serviced by *list scheduling* —
when a reservation of a job is active, it can execute any
ready node of the DAG job. List scheduling is work-conserving
— namely, at every point in time in which the DAG job has
pending workload and the system provides service, some node
is executing.

As long as the assigned server budgets are sufficient for
DAG tasks to meet their deadlines, the system can use any
scheduling algorithm and schedulability test to schedule
these sequential servers on a homogeneous multiprocessor
system with \( M \) processors. The exact time of service, i.e., the
schedule of the reservation servers, is determined by the
applied multiprocessor scheduling algorithm. Therefore,
under reservation-based federated scheduling the problem of
scheduling DAG task sets and the analysis thereof is divided
into the following two problems:

1) Scheduling of sporadic constrained- or arbitrary-deadline
reservation servers to satisfy the budget requirement.

2) Assigning provably sufficient budget to service an arbi-
trary DAG task.

The reservation concept is illustrated in Figure 1b, where
two reservations are partitioned on two different processors
and scheduled according to an arbitrary scheduling algorithm.
The two reservation servers provide 7.5 time-units of runtime
budget (as computed by the \textit{R-MIN} algorithm, explained later)
over the interval \([0,9]\) in parallel to service the DAG task
shown in Figure 1a. To improve readability, higher priority
tasks or reservation servers in the system are not included in
Figure 1b. Instead, the resulting preemptions and the servicing
of the reservations are denoted by the different hatchings.

If a reservation is scheduled to execute but no subjob of the

served DAG task is eligible, the reservation spins. For instance at time $t = 5.5$ on the second processor, no subjob is eligible to be serviced, due to unmet precedence constraints. In this case, the associated scheduled reservation spins. Note that the total reservation of 15 necessarily exceeds the work of the DAG job 10 since the reservation must guarantee that the job completes regardless of how the reservation servers are scheduled by the underlying scheduling algorithm.

**B. Sufficient Conditions for Reservation Server Design**

To guarantee the schedulability of a DAG task within its reservation servers, we must quantify the processor time budgets that are always sufficient to schedule a DAG task over an interval. Here, we derive conditions for designing reservation servers for DAG tasks. In particular, we will show that any design that satisfies the two conditions in the following theorem guarantees that the DAG job is completed by its deadline.

**Theorem 1:** Suppose that $m_i$ sequential reservation servers with budgets $E_{i,1}, E_{i,2}, \ldots, E_{i,m_i}$ are assigned to serve a DAG task $\tau_i$. The job of task $\tau_i$ arrived at time $t_0$ can be finished no later than its absolute deadline $t_0 + D_i$ if the two following conditions are satisfied:

- **Schedulability Condition:** The $m_i$ reservation servers can be guaranteed to finish no later than their absolute deadline at $t_0 + D_i$;

- **Reservation Condition:** $C_i + L_i \cdot (m_i - 1) \leq \sum_{j=1}^{m_i} E_{i,j}$.

**Proof:** We consider an arbitrary execution schedule $S$ of the $m_i$ servers to complete their budgets from $t_0$ to $t_0 + D_i$. Suppose, for contradiction, that the schedulability and reservation conditions hold but that the DAG job of task $\tau_i$ released at time $t_0$ misses its deadline at time $t_0 + D_i$ in the schedule $S$.

Since the list scheduling algorithm is applied, the DAG job is executed within the reservation servers in a work-conserving manner — at each time step in $S$ each executing server will process one ready node of the DAG job. Therefore, if there is some server idling at a time step, all the ready nodes must be processed by some of the executing servers.

During $S$, we define a time step as a complete step if the number of servers executing is no more than the number of ready nodes. Hence, in these complete steps all the executing servers are processing useful work of the job. In contrast, we define a time step is incomplete if the number of executing servers is more than the number of ready nodes, i.e., there is at least one executing server idling. We denote the number of incomplete steps during the interval from $t_0$ to $t_0 + D_i$ as $x$.

Since the job is not finished at $t_0 + D_i$, there is at least one ready node at any time step. By the definition of incomplete time steps and critical-path length, if $x > L_i$, in these incomplete steps the job must have finished all of its nodes, which leads to contradiction. Thus, it must be the case that $x \leq L_i$. In the worst case, i.e., solely sequential execution, in each of the incomplete step all the $m_i$ servers are executing and $m_i - 1$ servers are idling. Since $x \leq L_i$, we know that the amount of server idling time is at most $(m_i - 1)x \leq (m_i - 1)L_i$.

Note that by the schedulability condition, the total amount of execution budgets the reservation servers get in the schedule $S$ is $\sum_{j=1}^{m_i} E_{i,j}$. Among them, only at most $(m_i - 1)L_i$ are idle. Therefore, the amount of useful work these servers complete is at least $\sum_{j=1}^{m_i} E_{i,j} - (m_i - 1)L_i$.

By the reservation condition, we know that $\sum_{j=1}^{m_i} E_{i,j} - (m_i - 1)L_i \geq C_i$. This means that the reservation servers process more work than the worst-case execution time of the job, which leads to contradiction. Therefore, the job must have completed by its deadline.

Note that in the above theorem the reservation server design is independent of the actual structure of the DAG job — it depends only on the worst-case work $C_i$ and critical-path length $L_i$ of the job. The following lemma argues that providing a reservation server with $E_{i,j} < L_i$ is never useful.

**Lemma 2:** Let $E_{i,1}, E_{i,2}, \ldots, E_{i,m_i}$ denote a set of sufficient reservation budgets for a DAG task $\tau_i$, i.e., $\sum_{j=1}^{m_i} E_{i,j} \geq C_i + L_i (m_i - 1)$. If there exists a reservation server with budget $E_{i,j} < L_i$, then removing this reservation already provides sufficient budget with one less reservation server and less cumulative budget.
Proof: By assumption, we know that \( \sum_{j=1}^{m_i} E_{i,j} - E_{i,j^*} + E_{i,j^*} \geq C_i + L_i(m_i - 1) \). Simple arithmetic yields \( \sum_{j=1}^{m_i} E_{i,j} - E_{i,j^*} + \geq C_i + L_i(m_i - 1) - E_{i,j^*} \). By the property that \( E_{i,j^*} < L_i \), it must be that \( \sum_{j=1}^{m_i} E_{i,j} - E_{i,j^*} + \geq C_i + L_i(m_i - 1) - E_{i,j^*} > C_i + L_i(m_i - 2) \).

From now on, we will implicitly assume that \( E_{i,j} \geq L_i \forall j \) due to Lemma 2. In particular, we define the stretch ratio \( \gamma_{i,j} \) for each reservation server and let \( E_{i,j} = \gamma_{i,j} L_i \) with \( 1 < \gamma_{i,j} \leq \frac{C_i}{L_i} \). Subsequently, we have the following corollary.

**Corollary 3:** Suppose that \( m_i \) sequential reservation servers with budgets \( \{\gamma_{i,1} L_i, \gamma_{i,2} L_i, \ldots, \gamma_{i,m_i} L_i\} \) are assigned to serve a DAG task \( \tau_i \). Task \( \tau_i \) is feasible if the budget assignment satisfies:

\[
L_i \cdot (m_i - 1) + C_i \leq \sum_{j=1}^{m_i} \gamma_{i,j} \cdot L_i \tag{1a}
\]

\[
\gamma_{i,j} \cdot L_i \leq D_i \quad \forall 1 \leq j \leq m_i \tag{1b}
\]

\[
\gamma_{i,j} > 1 \quad \forall 1 \leq j \leq m_i \tag{1c}
\]

Eq. (1) has the following properties. First, the equation depends only on the task parameters \( C_i, L_i, D_i \) as well as the number of reservation servers \( m_i \). Second, as \( m_i \) increases, the sum of execution requirements \( \sum_{j=1}^{m_i} \gamma_{i,j} \cdot L_i \) also increases. Therefore, there are many feasible designs of reservation servers for the same DAG task. The best reservation assignment depends on the scheduling algorithm for the reservation servers and the corresponding analysis used to verify schedulability of the servers.

V. RESERVATION SERVER ASSIGNMENT ALGORITHMS

In this section, we describe two simple algorithms for calculating reservation budgets for DAG tasks which both have advantages and disadvantages. The first algorithm is the direct generalization of federated scheduling core allocation. The second algorithm is designed so that we can easily prove the speedup bounds for the reservation-based federated scheduling in Sections VI and VII.

In principle, from Theorem 1 and Corollary 3 we know that the reservation servers of the same task \( \tau_i \) could potentially have different budgets \( E_{i,j} \) and stretch ratio \( \gamma_{i,j} \). However, in the two simple server assignment algorithms, we compute the reservation budgets by enforcing equal-reservation \( E_i \) and equal stretch ratio \( \gamma_i \) for task \( \tau_i \). Therefore, the conditions for feasible reservation servers can be solved analytically to \( L_i \cdot (m_i - 1) + C_i \leq \gamma_i \cdot m_i \cdot L_i \), which yields \( \gamma_i \leq \frac{C_i - L_i}{L_i \cdot (\gamma_i - 1)} \leq m_i \). Note that the notation of \( \gamma_{i,j} \) from now on changed to \( \gamma_i \) due to the equality of all \( m_i \) reservation servers.

Since the number of reservation servers must be a natural number, we know that the smallest number of reservation servers \( m_i \) required under the equal-reservation constraint given a stretch ratio \( \gamma_i \) is

\[
m_i = \left\lceil \frac{C_i - L_i}{L_i \cdot (\gamma_i - 1)} \right\rceil \tag{2}
\]

A. R-MIN: Minimum Reservation Budgets

The intuition behind the first algorithm, namely R-MIN, is similar to the original federated scheduling core allocation, which is to assign the minimum reservation budgets that can still guaranteed the schedulability of tasks. Specifically, R-MIN classifies tasks into light and heavy tasks based on whether a task requires to be serviced by more than one reservation server or not. Based on the classification, the algorithm assigns each light task one reservation server with a budget identical to the work of the task. It assigns each heavy task the minimum number of reservation servers \( m_i = \left\lceil \frac{C_i - L_i}{L_i \cdot (\gamma_i - 1)} \right\rceil \) using the equal-reservation constraint. The servers of the same task is assigned the minimum stretch ratio \( \gamma_i = 1 + \frac{C_i - L_i}{m_i L_i} \), i.e., the minimum budget, while guaranteeing the schedulability of the task.

Therefore, the R-MIN has the following properties:

**Theorem 4:** The R-MIN algorithm generates a minimal number of sporadic equal-reservation servers with minimum budgets for a given sporadic constrained- or arbitrary-deadline DAG task set that provide sufficient resources to service their respective DAG tasks.

Proof: First, the minimum \( m_i \) gives the minimum total execution requirements \( \sum_{j=1}^{m_i} \gamma_i \cdot L_i \) in inequality (1a). In equal-reservations, the left-hand side of the above equation (2) is minimized if \( \gamma_i \) is maximized, i.e., \( m_i = \left\lceil \frac{C_i - L_i}{L_i \cdot (\gamma_i - 1)} \right\rceil \), and the corresponding smallest \( \gamma_i \) that achieves an equally minimal number of reservations is given by \( 1 + \frac{C_i - L_i}{m_i L_i} \).

The R-MIN strategy is a generalized approach of federated scheduling in the sense that an instance of R-MIN can be transformed into an instance of federated scheduling by inflating the assigned reservation budgets \( E_i \) to \( T_i \) (for implicit-deadline systems) such that each heavy DAG task is reserved 100% processor utilization exclusively. One can consider R-MIN as a resource optimized version of federated scheduling.

The R-MIN strategy assigns the minimum total budgets to all the tasks, leaving the minimum idling time inside the reservation servers. On the other hand, this strategy is the closest to federated scheduling and inherits its disadvantages. First, each task’s reservation is calculated completely independently from all the other jobs in the system. Second, each reservation may be “tight”, i.e., \( E_{i,j} \) may be very close to \( D_i \). Both these properties may make it difficult to ensure that the assigned servers of a task set are schedulable even though individual tasks require their minimum budgets. This strategy is unlikely to provide a constant speedup bound for the same reason that federated scheduling does not admit a constant speedup bound for constrained-deadline tasks. Therefore, we will now look at a different strategy which potentially increases the number of reservation servers for each task, but each reservation server may have a smaller execution requirement compared to R-MIN.

B. R-EQUAL: Equal Stretch Ratio for All Tasks

As mentioned above, one of the reasons R-MIN may generate unschedulable reservation servers is that each job calculates its own stretch ratio which can be as large as possible for that job. Jobs that have large stretch ratio \( \gamma_i \) have little slack, i.e., their reservation \( E_i = \gamma_i L_i \) is very close to \( D_i \). We now present an algorithm R-EQUAL that uses a single common stretch ratio \( \gamma \) across all tasks in Algorithm 1.
In algorithm R-EQUAL, all DAG tasks are classified into heavy and light tasks based on whether \( C_i > \gamma L_i \) for a single common stretch ratio \( \gamma \), i.e., whether they require more than one reservation (as generated by the algorithm) to be feasibly serviceable or not based on \( \gamma \). Since \( \gamma \) must be valid for all DAG tasks, i.e., \( L_i < \gamma L_i \leq D_i \), we know that \( 1 < \gamma \leq \min_i \{ \frac{D_i}{L_i} \} \). Based on the classification, the algorithm assigns each light task one reservation server with the same budget as the work of the task. It assigns each heavy task \( m_i = \left\lfloor \frac{C_i - L_i}{(\gamma - 1) L_i} \right\rfloor \) reservation servers given the same stretch ratio \( \gamma \).

It turns out that R-EQUAL provides good analytical results and allows us to prove speedup bounds. It does not, however, lead to heuristically good reservations as will be seen in the evaluations. Thus, in Section VIII we present an improved server assignment algorithm that can exploit the properties of the scheduling algorithm applied to the servers.

In the following theorem, we prove an important property of R-EQUAL servers, which is used to prove the speedup bounds for reservation-based federated scheduling.

**Theorem 5**: Suppose that \( \gamma > 1 \) is given and there are exactly \( m_i \) reservation servers for task \( \tau_i \) with \( m_i \geq 2 \). If \( C_i = \sum_{j=1}^{m_i} E_{i,j} = C_i + (m_i - 1)\cdot L_i \) and \( \gamma = \frac{\sum_{j=1}^{m_i} E_{i,j}/L_i}{\sum_{j=1}^{m_i} E_{i,j}} \), then \( C_i' \leq (1 + \frac{1}{\gamma - 1}) \cdot C_i \).

*Proof:* By the assumption \( L_i > 0 \) and \( \gamma > 1 \), the setting of \( m_i \) implies that
\[
m_i - 1 < \frac{C_i - L_i}{L_i(\gamma - 1)} \leq m_i \quad (3)
\]
\[
\Rightarrow (m_i - 1)(\gamma - 1)L_i < C_i - L_i \leq m_i L_i(\gamma - 1) \quad (4)
\]
\[
\Rightarrow C_i + (m_i - 1)L_i \leq m_i \gamma L_i < C_i + (m_i + \gamma - 2)L_i \quad (5)
\]

Because \( \gamma > 0 \), Eq. (5) implies \( (m_i - 1)L_i < \frac{C_i + (m_i - 2)L_i}{\gamma} \).

Since \( C_i' = C_i + (m_i - 1)L_i \) by definition, we know
\[
C_i' < C_i + \frac{C_i + (m_i - 2)L_i}{\gamma}
\]
\[
= \frac{C_i (\gamma + 1)}{\gamma} + \frac{(m_i - 2)L_i}{\gamma} \cdot \frac{C_i}{(m_i - 1)(\gamma - 1)}
\]
\[
\leq C_i \left( \frac{\gamma + 1}{\gamma} + \frac{1}{\gamma^2 - \gamma} \right)
\]
\[
= \left( 1 + \frac{1}{\gamma - 1} \right) \cdot C_i
\]
where \( <_1 \) is due to \( L_i < \frac{C_i}{m_i - 1}(\gamma - 1) + 1 < \frac{C_i}{m_i - 1}(1 - 1) \) by reorganizing the condition in Eq. (4) and \( \leq_2 \) is due to \( m_i \geq 2 \) and \( \frac{m_i}{m_i - 1} \leq 1 \).

**VI. Partitioned Scheduling for Reservation Servers**

This section analyzes the theoretical properties of reservation-based federated scheduling when the reservation servers are scheduled under multiprocessor partitioned scheduling. Using the R-EQUAL algorithm and Theorem 5, a worst-case setting can be derived to prove a constant speedup factor of reservation-based federated scheduling with respect to an optimal DAG task scheduling algorithm.

The speedup bounds of different scheduling algorithms for ordinary sequential real-time tasks under partitioned scheduling. In particular, Baruah and Fisher developed a greedy heuristic in [9].

**Definition 2:** Under Deadline-Monotonic Partitioning (DMP), sequential tasks are considered in a non-decreasing order of their relative deadlines. When a task \( \tau_k \) is considered in this order, if task \( \tau_k \) and the other previously assigned tasks on a processor can be feasibly scheduled under the specified scheduling strategy, then task \( \tau_k \) is assigned to one of such processors (if there are more than one). Otherwise, task \( \tau_k \) is assigned to a newly allocated processor.

This strategy is further analyzed by Chen et al. in [17], [18] and proved to have a speedup bound of 3. Note that DMP only specifies the order of the tasks to be considered and assigned. The underlying uniprocessor scheduling strategy after task partitioning can be arbitrary, e.g., EDF or deadline-monotonic fixed-priority scheduling.

Since we consider DMP in this section, we index the tasks such that \( D_i \leq D_j \) if \( i \leq j \). Before we prove the competitiveness of reservation-based federated scheduling under DMP, we first briefly restate the schedulability tests used for the partitioned scheduling of arbitrary-deadline task sets, that are used in this paper for scheduling server tasks.

Suppose that the partitioning algorithm attempts to assign a server task \( \tau_k \) with budget \( E_k \) to a processor \( m \) and \( T_m \) is the set of higher-priority server tasks that are already assigned on processor \( m \). When considering arbitrary-deadline tasks on uniprocessor, Fisher, Baruah, and Baker [22] provided the following approximated schedulability test:

\[
E_k + \sum_{\tau_i \in T_m} \left( 1 + \frac{D_k}{T_i} \right) E_i \leq D_k \quad \text{and} \quad (6a)
\]
\[
U_k + \sum_{\tau_i \in T_m} U_i \leq 1 \quad (6b)
\]

Bini et al. [13] provided a tighter schedulability test as follows:

\[
E_k + D_k \left( \sum_{\tau_i \in T_m} U_i \right) + \sum_{\tau_i \in T_m} E_i - \sum_{\tau_i \in T_m} U_i E_i \leq D_k \quad (7a)
\]
\[
U_k + \sum_{\tau_i \in T_m} U_i \leq 1 \quad (7b)
\]
With these analyses, we now prove that DMP and reservation-based federated scheduling can yield constant speedup bounds. To prove the speedup bound, we incorporate the over-provisioning of the generated reservation servers with respect to the worst-case execution time of the original DAG tasks.

**Theorem 6:** A system of arbitrary-deadline DAG tasks scheduled by reservation-based federated scheduling under DMP, in which each processor uses deadline-monotonic fixed-priority scheduling for the reservation servers, has a constant speedup factor of $3 + 2\sqrt{2}$ with respect to any optimal scheduler by setting $\gamma$ to $1 + \sqrt{2}$.

**Proof:** We prove this by adopting $R$-EQUAL (c.f. Algorithm 1) with a setting of $\gamma = 1 + \sqrt{2}$. Suppose, for contradiction, that reservation-based federated scheduling under DMP is given a speedup of $\alpha > 3 + 2\sqrt{2}$ compared to the optimal scheduler but still fails to schedule the DAG tasks. By the definition of $R$-EQUAL and Corollary 3 we know that it fails to schedule the DAG tasks if and only if DMP cannot partition all the assigned reservation servers.

Since we apply DMP, we index the tasks such that $D_i \leq D_j$ if $i \leq j$. Suppose that $\tau_k,i$ is a reservation server assigned to the DAG task $\tau_k$ that is not able to be partitioned to any of the given $M$ processors, where $1 \leq \ell \leq m_k$. Let $M_1$ be the set of processors in which Eq. (6a) fails. Let $M_2$ be the set of processors in which Eq. (6a) succeeds but Eq. (6b) fails. Since $\tau_k,i$ cannot be assigned on any of the $M$ processors, we have $|M_1| + |M_2| = M$.

By the violation of Eq. (6a), we know that

$$|M_1| E_{k,i} + \sum_{m \in M_1} \sum_{i,j \in T_m} \left(1 + \frac{D_k}{T_i}\right) E_{i,j} > |M_1| D_k$$

$$\Rightarrow |M_1| E_{k,i} \frac{D_k}{T_k} + \sum_{m \in M_1} \sum_{i,j \in T_m} \left(\frac{E_{i,j}}{D_k} + \frac{E_{i,j}}{T_i}\right) > |M_1|$$ (8)

By the violation of Eq. (6b), we know that

$$|M_2| \frac{E_{k,i}}{T_k} + \sum_{m \in M_2} \sum_{i,j \in T_m} \frac{E_{i,j}}{T_{i,j}} > |M_2|$$ (9)

Due to the definition $\sum_{i,j=1}^m E_{i,j} = C'_i$, and the fact that $\tau_{i,j}$ is assigned either on a processor of $M_1$ or on a processor of $M_2$ if $\tau_{i,j}$ is assigned successfully prior to $\tau_k,i$, by taking the summation of Eqs. (8) and (9) we know that

$$M \frac{E_{k,i}}{ \min \{T_k, D_k\}} + \sum_{i=1}^k \left(\frac{C'_i}{T_i} + \frac{C'_i}{D_k}\right) > M$$ (10)

By the reservation-budget setting $E_{k,i} = \gamma L_k$ and by Theorem 5, the above inequality can be upper bounded by

$$M \gamma L_k \frac{\min \{T_k, D_k\}}{\min \{T_k, D_k\}} + \sum_{i=1}^k \left(\frac{C_i}{T_i} + \frac{C_i}{D_k}\right) \left(1 + \frac{1}{\gamma - 1}\right) > M$$ (11)

By assumption, the reservation-based federated scheduling under DMP is given a speedup of $\alpha > 3 + 2\sqrt{2}$ compared to the optimal scheduler. Since the DAG task set is schedulable under the optimal scheduler on unit speed processors, on $\alpha$-speed processors we have $\alpha L_i \leq \min \{T_k, D_k\}$, $\alpha \sum_{i=1}^k \frac{C_i}{T_i} \leq M$ and $\alpha \sum_{i=1}^k \frac{C_i}{D_k} \leq M$. Therefore, we have

$$\frac{1}{\alpha} \geq \max \left\{ \frac{L_k}{\min \{T_k, D_k\}}, \sum_{i=1}^k \frac{C_i}{MT_i}, \sum_{i=1}^k \frac{C_i}{MD_k} \right\}$$ (12)

$$\Rightarrow \frac{1}{\alpha} \geq \left(1 + \frac{1}{\gamma - 1}\right) \frac{1}{\alpha} > 1$$ (13)

$$\Rightarrow \frac{1}{\alpha} > 1 + \frac{1}{\gamma \sqrt{\frac{2}{\gamma^2 + \gamma - 1}} = \gamma - 1 - 2} = \frac{2\gamma + \gamma - 1}{\gamma^2 + \gamma - 1}$$ (14)

Hence, $\alpha < 3 + 2\sqrt{2}$ leads to contradiction. Therefore, the speedup factor is at most $3 + 2\sqrt{2}$. Note that the setting of $\gamma$ as $1 + \sqrt{2}$ is in fact to maximize $\frac{\gamma - 1}{\gamma^2 + \gamma - 1}$.

**Corollary 7:** Based on Theorem 6, the speedup factor of reservation-based federated scheduling under DMP with a parameter $\gamma > 1$ in the $R$-EQUAL algorithm is $\frac{\gamma + \gamma - 1}{\gamma^2 + \gamma - 1}$.

**Proof:** This follows from the proof in Theorem 6.

**Theorem 8:** A system of arbitrary-deadline DAG tasks scheduled by reservation-based federated scheduling under DMP, in which each processor uses EDF for the reservation servers, admits a constant speedup factor of $3 + 2\sqrt{2}$ with respect to any optimal scheduler by setting $\gamma$ to $1 + \sqrt{2}$.

**Proof:** Since EDF is optimal on uniprocessor and is no worse than deadline monotonic, the task partitioning algorithm and analysis used in Theorem 6 yields the result directly.

Note that the setting of $\alpha = 1 + \sqrt{2}$ is to achieve a bounded speedup factor in the analysis. Nevertheless, while ensuring the worst-case behaviour, such an enforcement is usually not beneficial for the average-case performance. Details regarding this matter can be found in a recent paper by Chen et al [19].

VII. GLOBAL SCHEDULING FOR RESERVATION SERVERS

In contrast to partitioned scheduling, global scheduling schemes employ a single ready queue and allow arbitrary task migrations. Analogously to the problem of computing reservation servers for partitioned scheduling, no single best server assignment algorithm exists for global scheduling. Instead, it depends on the analysis used to verify the schedulability of server tasks, since different schedulability analyses may be sensitive to different parameters. For example, if the schedulability test only considers the cumulative demands of the generated reservation servers, then the individual settings of the reservations become irrelevant and we should thus choose the one with minimal reservation demands as in $R$-MIN. Otherwise, optimization techniques that are suitable to optimize the parameters relevant for the schedulability analysis may be used, e.g., linear- or quadratic-programming in conjunction with the constraints for feasible reservation systems.

Global scheduling algorithms for constrained- and arbitrary-deadline sequential tasks have been extensively studied, e.g., [8], [10], [11], [23], [24], [40]. In general, any of these schedulability tests can be applied for validating the underlying global scheduling algorithm.

To prove that the speedup factor of global deadline-monotonic (DM) scheduling for arbitrary-deadline DAG task sets is the same as in partitioned deadline-monotonic scheduling, namely $3 + 2\sqrt{2}$, we adopt the following definitions.
The load of the server tasks $\mathbf{T}$ is defined as $\text{LOAD}(\mathbf{T}) \triangleq \max_{t > 0} \left\{ \sum_{\tau_k \in \mathbf{T}} \frac{\text{DBF}(\tau_k,t)}{t} \right\}$, where $\text{DBF}(\tau_k,t)$ is the demand-bound function of server task $\tau_k$ during interval length $t$ as introduced by Baruah [7]. Further, the maximum density of the server tasks is defined as $\delta_{\text{max}}(\mathbf{T}) \triangleq \max_{\tau_k \in \mathbf{T}} \left\{ \frac{E_k}{\min\{D_k, T_k\}} \right\}$.

We use the following sufficient (but more pessimistic) schedulability test derived by Chen et al. (Theorem 4.7 in [20]) for scheduling reservation servers under global DM scheduling.

**Theorem 9:** A sequential arbitrary-deadline task set $\mathbf{T}$ is schedulable by global deadline-monotonic scheduling on $M \geq 1$ processors if

$$2 \cdot \text{LOAD}(\mathbf{T}) + (M - 1) \cdot \delta_{\text{max}}(\mathbf{T}) \leq M$$

**Proof:** By Theorem 4.7 in Chen et al. [20], a task set of sporadic arbitrary-deadline tasks that is indexed according to global deadline-monotonic priority assignment is schedulable on $M$ processors if for all tasks $\tau_k$

$$\delta_k + \sum_{i=1}^{k-1} \left( \frac{C_i - C_i U_i}{D_k} + U_i \right) \leq M - (M - 1) \cdot \max\{\delta_i, U_i\},$$

Further, by Corollary 5.1 [20]

$$\delta_k + \sum_{i=1}^{k-1} \left( \frac{C_i - C_i U_i}{D_k} + U_i \right) \leq 2 \cdot \text{LOAD}(k)$$

and due to the fact that $\max\{\delta_i, U_i\} \leq \delta_{\text{max}}(k)$ (since $\delta_i \geq U_i$), we obtain the over-approximated sufficient schedulability test stated in Theorem 9.

This sufficient condition is used to prove the following theorem:

**Theorem 10:** A system of arbitrary-deadline DAG tasks scheduled by reservation-based federated scheduling using global deadline-monotonic scheduling to schedule the generated reservation servers has a constant speedup factor of $3 + 2\sqrt{2}$ with respect to any optimal scheduler.

**Proof:** We prove this by adopting the $R$-EQUAL algorithm (Alg. 1) with $\gamma = 1 + \sqrt{2}$. Similar to the proof of Theorem 6, for contradiction we assume that the algorithm is given a speedup of $\alpha > 3 + 2\sqrt{2}$ compared to the optimal scheduler but still fails to schedule the DAG tasks, which is because global DM cannot schedule all the assigned reservation servers.

Let $\mathbf{T}$ denote the original set of arbitrary-deadline DAG tasks and let $\mathbf{T}'$ denote the set of associated reservation servers that are generated by $R$-EQUAL. Since $\mathbf{T}'$ fail to suffice theorem Thm. 10, we have

$$2 \cdot \text{LOAD}(\mathbf{T}') + (M - 1) \cdot \delta_{\text{max}}(\mathbf{T}') > M \Rightarrow$$

$$2 \max_{t > 0} \left\{ \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}(\tau_i,t)}{t} \right\} + \frac{(M - 1)}{M} \cdot \max_{\tau_i \in \mathbf{T}'} \left\{ \frac{E_i}{\min\{D_i, T_i\}} \right\} > 1$$

(19)

Since $E_i = \gamma L_i$ and by theorem Thm. 5, it follows that

$$2(1 + \frac{1}{\gamma - 1}) \max_{t > 0} \left\{ \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}(\tau_i,t)}{t} \right\} + \frac{(M - 1)}{M} \cdot \gamma \cdot \max_{\tau_i \in \mathbf{T}'} \left\{ \frac{L_i}{\min\{D_i, T_i\}} \right\} > 1$$

(20)

Again, by assumption the reservation-based federated scheduling under global DM is given a speedup of $\alpha > 3 + 2\sqrt{2}$ compared to the optimal scheduler. Since the DAG task set is schedulable under the optimal scheduler on unit speed processors, on $\alpha$-speed processors we have $\alpha L_i \leq \min\{T_k, D_k\}$ and $\alpha \cdot \max_{t > 0} \left\{ \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}(\tau_i,t)}{t} \right\} \leq M$. Therefore, we have

$$3 + 2\sqrt{2} \geq \gamma^2 + \gamma \geq 2\left(1 + \frac{1}{\gamma - 1}\right) + \gamma \cdot (1 - \frac{1}{M}) > \alpha$$

Hence, $\alpha < 3 + 2\sqrt{2}$ leads to contradiction. As a result, the speedup factor is at most $3 + 2\sqrt{2}$.

**VIII. IMPROVED ALGORITHMS FOR RESERVATION SERVERS**

**Algorithm 2** The SOF algorithm yields feasible partitions and associated reservations.

**Require:** An arbitrary-deadline DAG task set $\mathbf{T}$, processors $M$, boundaries $b_i, b_2, \ldots, b_N$.

**Ensure:** Feasible partition and reservations, that can service $\mathbf{T}$ provably (if one could be found).

1. $T_H, T_L \leftarrow \text{MIN}(\mathbf{T})$ or $R$-EQUAL$(\mathbf{T})$
2. re-index $T_H \cup T_L$ such that $D_i \leq D_j$ for $i < j$
3. for each $\tau_{i,j}$ in $T_H \cup T_L$ do
4. if $\tau_{i,j}$ in $T_L$ then
5. if partition by preference failed then
6. return Partition Failure
7. else
8. Failure $\leftarrow$ True
9. $\ell_i$ is no more than $\max\left\{ \left\lfloor \frac{C_i}{L_i} \right\rfloor, \left\lfloor \frac{C_i - L_i}{\ell_i(\gamma - 1)} \right\rfloor, b_i \right\}$
10. if partition by preference failed then
11. revoke all already partitioned $\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,j}$ belonging to DAG task $\tau_i$
12. $\ell_i \leftarrow \ell_i + 1$ (increment the number of reservation servers and re-try)
13. for each $j$ in $\{1, 2, \ldots, \ell_i\}$ do
14. $E_{i,j} \leftarrow \frac{C_i}{L_i} + (1 - \frac{1}{\ell_i}) \cdot L_i$ (compute evolved reservation budgets)
15. end for
16. continue
17. else
18. Partition $\tau_{i,j}$ to processor as chosen by preference
19. Failure $\leftarrow$ False
20. break
21. end if
22. end while
23. if Failure is True then
24. return Partition Failure
25. end if
26. end if
27. end for
28. end for
29. return Partition and Reservations

In this section, we consider a class of algorithms where reservation server assignment is adapted based on whether the initially assigned servers are schedulable by an applied scheduling algorithm. In principle, reservation-based federated scheduling under any suitable multiprocessor scheduling algorithm for servers can be improved using this strategy. Here, we take DMP as an example, highlight one such adaptive approach and analyze its speedup bound.

We present the Split-On-Fail (SOF) algorithm in Alg. 2. The SOF algorithm attempts to partition all generated reservation servers according to a schedulability test for arbitrary-deadline
tasks by preference, e.g., worst-fit, first-fit, and best-fit. In order to reduce the algorithmic complexity, the partitioning of each group of servers that belong to the same DAG task is done on the premise that all previously partitioned groups are already feasibly partitioned and their settings do not change. Additionally, light tasks are excluded from the adaptation process. Whenever a reservation server cannot be partitioned, an additional reservation server is added to the group and the individual reservation-budgets are decreased appropriately.

By the improved algorithm, the set of all possible reservation budgets is reduced from a theoretical feasible interval $L_i < E_i < D_i$. The feasible budget sizes can be calculated given the number of servers $\ell_i$ using $E_i = \frac{C_i}{L_i} + (1 - \frac{1}{\ell_i}) \cdot L_i$.

The intention is to improve schedulability by increasing the number of reservation servers, whilst decreasing their individual budgets. Note that any priority policy that assigns equal priorities to all reservations belonging to the same group will result in non-equal reservations. This is due to the fact that if multiple reservations with the same priority are partitioned onto the same processor, this behaves as if there was only one reservation with the individual reservation budgets accumulated on that processor. Furthermore, note that since the reservation-budgets are determined upon partitioning, different partitioning strategies, i.e., best-fit, worst-fit, and first-fit, result in different reservation-budgets effectively.

By using $R\text{-EQUAL}$ with $\gamma = \min_{i<k} \left\{ \frac{D_i}{T_i} \right\}$ to generate the initial reservations and for any $b_i > 1 \in \mathbb{N}$, we prove the following theorem assuming that $\gamma > 1$.

**Theorem 11:** The speedup factor of SOF under Deadline-Monotonic Partitioning (DMP), in which each processor uses EDF or DM for the reservation servers, is at most $\min\left\{ \frac{\gamma^2 + \gamma}{\gamma - 1}, 4 - \min\left\{ \frac{L_k}{C_k}, \frac{\gamma L_k - L_k}{C_k - L_k}, \frac{1}{b_k - 1} \right\} \right\} + 2 \max_{i<k} \left\{ \max\left\{ \frac{C_i}{L_i}, \frac{C_i - L_i}{L_i}, b_k - 1 \right\} \right\}$.

**Proof:** If SOF cannot find a feasible partition and reservations, then the initial reservation-budgets cannot be partitioned feasibly either. Due to Corollary 7 and Theorem 8, this yields that the speedup factor is at most $\frac{\gamma^2 + \gamma}{\gamma - 1}$. We use the same arguments as in Theorem 6, including the definition of $\tau_k$ and the index of the tasks according to DMP, i.e., $D_i \leq D_k$ for $i = 1, 2, \ldots, k - 1$. For this case, we have

$$\left(2 - \frac{1}{c_k}\right) \cdot \min\left\{ \frac{L_k}{T_k}, L_k \right\} + \sum_{i=1}^{k} \frac{C_i}{MT_i} + \frac{C_i}{MD_k} + \sum_{i=1}^{k} (c_i - 1) \cdot \left( \frac{L_i}{MT_i} + \frac{L_i}{MD_k} \right) > 1$$

Where due to the definition of the boundaries $c_i = \max\left\{ \frac{C_i}{T_i}, \frac{C_i - L_i}{L_i}, \frac{1}{b_i} \right\}$, it must be that $c_i \geq \frac{C_i}{T_i} \geq \frac{C_i}{L_i}$.

Let $\alpha = \frac{2 - \frac{1}{c_k} + 2 \cdot \max_{i<k} \{c_i - 1\}}{\min\{1, \frac{L_k}{T_k}, \sum_{i=1}^{k} \frac{C_i}{MT_i}, \sum_{i=1}^{k} \frac{C_i}{MD_k}\}}$ denote a necessary condition for schedulability. Then, the following inequality holds:

$$4 - \frac{1}{c_k} + 2 \cdot \max_{i<k} \{c_i - 1\} > \frac{1}{\alpha} \quad (21)$$

Since,

$$c_i - 1 = \max\left\{ \frac{C_i}{L_i}, \frac{C_i - L_i}{L_i(\gamma - 1)}, b_i \right\} - 1 \leq \max\left\{ \frac{C_i}{L_i}, \frac{C_i - L_i}{L_i(\gamma - 1)}, b_i \right\} - 1$$

holds, the equation can be reformulated to

$$4 - \min\left\{ \frac{L_k}{C_k}, \frac{\gamma L_k - L_k}{C_k - L_k}, \frac{1}{b_k - 1} \right\} + 2 \cdot \max_{i<k} \left\{ \max\left\{ \frac{C_i}{L_i}, \frac{C_i - L_i}{L_i}, b_k - 1 \right\} \right\}$$

This proves the theorem.

**IX. EXPERIMENTAL EVALUATION**

This section presents the evaluation results. We first numerically evaluate the reservation-based federated scheduling under partitioned schedulers (as our optimization for reservation server allocation is mainly for partitioned schedulers) using synthetic task sets. We then empirically evaluate the reservation-based federated scheduling under the global DM scheduler (as there is no budget-based partitioned scheduler in the RTCG framework [32]) implemented on a real multicore platform.

**A. Evaluations Based on Synthetic Task Sets**

**Task Set Generation:** We used the Randfixedsum algorithm [21] to generate utilizations for task sets of fixed size with individual task utilizations $0 < U_i \leq M$, where $M$ denotes the number of processors. Each task was characterized by $\tau_i = (C_i, L_i, D_i, T_i)$, where the periods were drawn uniformly from $(0, 100)$ to cover a wide range of characteristics. The range of the parameters $\alpha = \frac{D_i}{T_i}$ and $\beta = \frac{L_i}{T_i}$ for generating deadline and critical-path length is changed for different settings.

We evaluate task sets with implicit, constrained, and arbitrary deadlines. For each setting, task sets for 8, 16, and 32 processor systems were generated. For each normalized utilization (in five percent steps), 100 task sets were generated, where each task set consists of 20 DAG tasks. This task set size was selected to be large enough to allow smaller individual utilization and thus more options for the partitioning, whilst being small enough to still be difficult to partition.

In the following, the proposed reservation-based federated scheduling algorithms, namely SOF in the $R\text{-MIN}$ and $R\text{-EQUAL}$ variants as well as the different partitioning heuristics first-fit, best-fit, and worst-fit were compared against the semi-federated scheduling approach by Jiang et al. [26]. Since the evaluations of Jiang et al. [26] did not suggest a notable difference between the performance of their two proposed algorithms, only the algorithm $SF[X + 1]$ was adopted and is referred to as $S\text{-FED}$. In semi-federated scheduling, the resource waste of federated scheduling is addressed by computing the required processors more tightly. That is, if federated scheduling requires at least $m_i \geq x$ many processors with $m_i \in \mathbb{N}$, then federated scheduling will allocate $\lfloor x \rfloor = (x + \epsilon)$-many processors for this task. This may lead to at most 200% of the required resources since $2 \geq \frac{c_k}{x + \epsilon}$, $x \geq 1$ for $\lim \epsilon \to 0$. Conversely, the proposed improvements decrease
with the number of required processors. To allow a fair evaluation and to make use of the advantage of the sporadic task model for the reservations, a SOF variant with partitioned EDF for arbitrary-deadlines using the linear demand-bound function approximation as proposed by Baruah [3] was adopted. Consequently, a scheduling test consisting of the reservation initialization method R-MIN or R-EQUAL, the uniprocessor scheduling algorithm EDF or DM, and a packing heuristic, results in 12 variants. Due to similar performance in the evaluations, only a subset of the variants that are representative of the group’s performance is illustrated.

**Experimental Results on Implicit-Deadline Task Sets:** The first set of experiments is shown in Fig. 2. It can be seen that for tasks with implicit deadlines S-FED and the best performing variant SOF-EDF-BF-MIN behave similarly up to a cutoff utilization of 60%, 45%, and 35% for 8, 16, and
The variants SOF-DM-BF-MIN and BF-MIN algorithm exhibits a substantially lowered cutoff utilization. This is due to the fact that R-EQUAL generates more reservations initially in comparison to R-MIN variants and thus decreases schedulability.

**Experimental Results on Constrained-Deadline Task Sets:**
In Fig. 3, the deadlines of constrained-deadline tasks were chosen uniformly from \((0.1T_i, T_i]\). Here the variants SOF-DM-BF-MIN and SOF-EDF-BF-MIN slightly outperform S-FED for 8, 16, and 32 processors, whereas the gap decreases with the number of processors. The S-FED is especially sensitive to small \(L_i/D_i\) and \(D_i/T_i\) ratios, and performs much worse, which is demonstrated in an extreme-case shown in Fig. 4.

**Experimental Results on Arbitrary-Deadline Task Sets:**
Finally, Fig. 5 presents arbitrary-deadline task sets where the deadline was chosen uniformly from \((0.1T_i, 10T_i]\). Since semi-federated scheduling does not support arbitrary-deadline DAG task sets, only SOF variants are evaluated. All SOF variants accept all task sets until 50\%, 20\%, and 10\% cutoff utilizations in 8, 16, and 32 processor platforms, respectively. The variants SOF-DM-BF-MIN and SOF-EDF-BF-EQ exhibit the best performance and behave identically. Additionally, we observe that SOF-DM-WF-MIN demonstrates the worst performance.

### B. Empirical Evaluation

We now describe our empirical results of reservation-based federated scheduling using it with DM scheduling for servers.

**Platform Implementation:** We implement a prototype platform that supports reservation-based federated scheduling under global DM by modifying an existing federated scheduling platform for OpenMP programs, namely RTCG in [32]. The main differences between the reservation-based federated scheduling under the global DM and the original federated scheduling include: (1) deadline monotonic priorities to the parallel threads of a task, (2) different numbers of threads generated for a task, where each thread corresponds to one reservation server in reservation-based federated scheduling in contrast to one dedicated core in the original federated scheduling, and (3) global execution of the threads of a task, instead of dedicated core assignment using federated scheduling.

We modify RTCG correspondingly to address these differences.

Experiments were conducted on a 48-core machine composed of 4 AMD Opteron 6168 processors. We reserved one processor, i.e., 12 cores, for system tasks, leaving 36 experimental processing cores. Based on this hardware specification, we ran the single socket 12-core experiments and multi-socket 36-core experiments. Linux with the PREEMPT_RT kernel patch was used as the underlying RTOS.

**Task Set Generation:** We use the similar approach as in Section IX-A to randomly generate synthetic task sets. In these experiments, all units are expressed in millisecond (ms). In addition, we randomly selected task period from \([4\text{ms}, 8\text{ms}, 16\text{ms}, 32\text{ms}, 64\text{ms}, 128\text{ms}]\) to form task sets with harmonic periods. For all the experiments, each task set was run for 100 hyper-periods.

We compare reservation-based federated scheduling under global DM in the R-MIN and R-EQUAL variants (namely GDM-MIN and GDM-EQ, respectively) against our implementation of global DM without any reservation (GDM-ALL). In addition, for experiments with implicit deadlines, we also compare against the original federated scheduling, i.e., RTCG platform (FS). To have a fair comparison, for experiments with constrained and arbitrary deadlines we modify the federated scheduling to use the \(\min \{D_i, T_i\}\) instead of \(D_i\) to calculate the core allocation for each task.

**Experimental Results:** The experiments on 12 cores are shown in Fig. 6. In all settings, GDM-MIN outperforms FS and FS is comparable or better than GDM-EQ. As discussed in Section VII, reservation-based federated scheduling under global DM should choose reservation servers with the minimum reservation demands, i.e., GDM-MIN, instead of other alternatives such as GDM-EQ. Additionally, GDM-MIN outperforms FS since without the dedicated core allocation a core will not idle as long as there are some unfinished jobs. Therefore,
GDM-MIN can more efficiently utilize the multicore platform. Finally, GDM-ALL performs significantly worse, which shows the necessity of calculating the proper reservation demands.

In addition to the experiments on 12 cores, the results for the same experimental settings on 36 cores are shown in Fig. 7. The performance trends of different scheduling algorithms on 36 cores are similar to those on 12 cores. It can be seen that GDM-MIN outperforms all other scheduling algorithms in terms of acceptance ratios.

X. Conclusion and Future Research

In this work, we show that reservation-based federated scheduling is competitive with the state-of-the-art of sporadic DAG task scheduling for the parametric model. Especially in the constrained-deadline and arbitrary-deadline cases, the reservation-based federated scheduling demonstrates good performance as well as robustness with respect to varying parameters. Since reservation-based federated scheduling can be used in conjunction with any pre-existing scheduling algorithm that supports sporadic sequential tasks, we believe that it has a great potential to be applied in practical systems even though it has a slight performance loss for certain scenarios.

For future work, we will consider the memory and bus contention, as this has been either ignored or pessimistically overestimated in the literature. For constrained-deadline sporadic task systems (without DAG) under fixed-priority scheduling, there have been a few recent research results, e.g., [1], [25]. We plan to explore the impact of such resource contention when designing reservation-based federated scheduling.

ACKNOWLEDGMENT

This paper is supported by DFG, as part of the Collaborative Research Center SFB876, project B2 (http://sfb876.tu-dortmund.de/), by NSF Grants CCF-1337218 and AITF-1733873, and by NJIT Seed Grant. This research was initiated in the Dagstuhl Seminar 17131 — Mixed Criticality on Multicore/Manycore Platforms.