I. Introduction

During the last two decades, multicores are more and more widely used in real-time systems to meet the rapidly increasing requirements in high performance computing and lowering the power consumption. To fully utilize the computational capacity of multicore processors, not only intertask parallelism, but also intratask parallelism need to be explored in the design and analysis of modern systems, where individual tasks are parallel programs and can potentially utilize more than one core at the same time during their executions. Parallel tasks are commonly supported by nowadays parallel programming languages, such as Cilk family [1], OpenMP [2], [3], and Intel’s Thread Building Blocks [4]. The primitives in these languages and libraries, such as parallel for-loops, omp task and fork/join or spawn/sync, results in intratask parallelism structures that can be well represented via graph-based task models. In the past few years, the real-time systems community has paid much attention to graph-based (parallel) task models, such as fork-join tasks [5], [6], synchronous tasks [7]–[11], and directed acyclic graph (DAG) tasks [12]–[25].

In this paper, we consider the general parallel tasks modeled as DAGs, where each vertex represents a sequence of instructions and each edge represents the interdependency constraints among the vertices. Real-time scheduling algorithms for DAG tasks can be classified into three paradigms: 1) decomposition-based scheduling [15], [17], [20], [22]; 2) global scheduling (without decomposition) [13], [16], [23]; and 3) federated scheduling [18], [26]–[29]. Decomposition-based scheduling first decomposes each DAG task into a set of sequential subtasks and assigns them intermediate release time and deadlines, and then schedules these sequential subtasks using a traditional multiprocessor scheduling policy for sequential tasks. In federated scheduling, the scheduler maintains a set of dedicated cores for each high-utilization task with utilization >1, and forces the remaining low-utilization task (with utilization ≤1) to be sequentially executed by the remaining (shared) cores.

This paper focuses on global scheduling, in particular, global earliest deadline first (GEDF) scheduling. Many existing systems, for example, Linux [30] and LITMUS [31] have provided efficient and scalable implementations of GEDF for sequential tasks, which suggests a potentially easy implementation for parallel tasks. However, schedulability analysis of GEDF for DAG tasks is a challenging problem. Theoretical work on real-time scheduling and schedulability analysis of real-time parallel tasks uses two quantitative metrics.
1) Resource Augmentation Bound (also called speedup factor) is a comparative metric with respect to some other (optimal) scheduler. A scheduler $S$ provides a resource augmentation bound of $\rho$ if it can successfully schedule any task set $\tau$ on $m$ cores of speed $\rho$ as long as the compared scheduler can schedule $\tau$ on $m$ cores of speed 1. A resource augmentation bound shows how close the performance of a scheduler is to the compared one, but it cannot be directly used as a schedulability test.

2) Capacity Augmentation Bound is an absolute metric that can be directly used for schedulability test. A scheduler $S$ has a capacity augmentation bound of $\rho$ if it can schedule any task set $\tau$ satisfying the following two conditions: a) the total utilization of $\tau$ is at most $m/\rho$ and b) the worst-case critical path length of each task is at most $1/\rho$ of its deadline. Capacity augmentation bounds are stronger than resource augmentation bounds in the sense that if a scheduler has a capacity augmentation bound of $\rho$, it is also guaranteed to have a resource augmentation bound of $\rho$. In parallel task scheduling, a capacity augmentation bound can serve as a simple linear-time schedulability test that requires no knowledge about the DAG structures except the critical path length and utilization of each task.

A. Contribution

In this paper, we derive the first capacity augmentation bound for GEDF scheduling of DAG tasks with constrained deadlines

$$\rho = \beta + 2\sqrt{\frac{\beta + 1}{m}}\left(\frac{1}{1 - \frac{1}{m}}\right)$$

(1)

where $m$ is the number of processing cores and $\beta$ is the maximal ratio of task period to deadline (see in Section III for a more formal definition). When $m$ becomes infinitely large, the bound approaches $\beta + 2\sqrt{\beta + 1}$. Moreover, we also prove that the capacity augmentation required by GEDF is at least $\beta + 2\sqrt{\beta + 1}/2$. Fig. 1 shows the figure of this capacity augmentation bound as a function of $\beta$.

There have been many previous works on both types of bounds for sporadic parallel tasks under different scheduling algorithms and different deadline constraints (see Section II for a review). To the best of our knowledge, the capacity augmentation bound for the problem setting considered in this paper is still open. It is worth mentioning that [13] introduced a simple schedulability test condition having the same time complexity and requiring the same information as our capacity augmentation bound. However, the test condition in [13] is more pessimistic than our capacity augmentation bound. We have conducted experiments to compare the acceptance ratio of these two tests, and the results show that our capacity augmentation bound significantly outperforms the test in [13] under different parameter settings.

The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the DAG task model and its runtime model. Section IV formally defines the notation and terminology related to the global EDF policy. Proofs of capacity augmentation bounds are presented in Section V. Evaluation result is shown in Section VI. Section VII gives concluding remarks.

II. RELATED WORK

The prior results on real-time scheduling and schedulability analysis of real-time parallel tasks can be classified into two categories: 1) those based on augmentation bound analysis and 2) those based on response time analysis (RTA).

A. Augmentation Bound Analysis

Augmentation bound analysis can be further classified as two subcategories: 1) resource augmentation bound and 2) capacity augmentation bound. Based on the resource bound, one can only propose a (pseudo-)polynomial time schedulability test with a bounded speedup, which cannot be directly applied on the platform with unit-speed cores. The capacity bound is the only theoretical quantitative metric that can serve as a sufficient schedulability test for the tasks on unit-speed cores. In the following we review previous work on resource augmentation bounds and capacity augmentation bounds for sporadic DAG task models with different deadline constraints (implicit, constrained, or arbitrary) under different scheduling algorithms (decomposition-based, global, and federated). The state-of-the-art results are summarized in Table I.

1) Resource Augmentation Bounds:

1) Decomposition-Based Scheduling: For decomposition-based scheduling, the associated resource augmentation bounds are indicated by their capacity augmentation bound results. Hence, we only survey the capacity augmentation bounds for decomposition-based scheduling in the next section.

2) Federated Strategy: For implicit-deadline DAG tasks, Li et al. [18] proved a resource augmentation bound of 2 with respect to hypothetical optimal scheduling algorithms. For constrained-deadline DAG tasks, Chen [24] showed that any federated scheduling algorithm has a resource augmentation bound of at least $\Omega(\min(m, n))$ with respect to any optimal scheduling algorithm, where $n$ is the number of tasks and $m$ is the
number of cores. With respect to any optimal federated scheduling algorithm, Baruah proved a speed-up factor of $3 - (1/m)$ for constrained deadline DAG tasks [26] and proved a speed-up factor of $4 - (2/m)$ for arbitrary deadline DAG tasks [27].

3) **Global Scheduling:** For a single recurrent DAG task with an arbitrary deadline, Baruah et al. [12] proved a bound of 2 under GEDF. For multiple DAG tasks with arbitrary deadlines, Li et al. [14] and Bonifaci et al. [13] proved a bound of $2 - (1/m)$ under GEDF, and Bonifaci et al. [13] proved a bound of $3 - (1/m)$ under deadline monotonic (DM) scheduling. All these bounds are with respect to an optimal scheduling algorithm.

2) **Capacity Augmentation Bounds:**

1) **Decomposition-Based scheduling:** The capacity augmentation bounds for decomposition-based scheduling are restricted to implicit-deadline DAG tasks. Earlier work began with synchronous tasks (a special case of DAG tasks). For a restricted set of synchronous tasks, Lakshmanan et al. [5] proved a bound of 3.42 using DM scheduling for decomposed tasks. For more general synchronous tasks, Saifullah et al. [7] proved a bound of 4 for GEDF and 5 for DM scheduling. For DAG tasks, Saifullah et al. [17] proved a bound of 4 under GEDF on decomposed tasks, and Jiang et al. [20] refined this bound to the range of $[2 - (1/m), 4 - (2/m)]$, depending on the DAG structure characteristics. For a special class of DAG task sets, Qamhieh et al. [22] proved a bound of $[(3 + \sqrt{5})/2]$. This is the best capacity augmentation bound known for task sets with multiple DAGs.

2) **Federated Strategy:** For multiple DAGs with implicit deadlines, Li et al. [18] proved a bound of 2 under federated scheduling. For mixed-criticality DAGs with implicit deadlines, Li et al. [29] proved that for high utilization tasks, the mixed criticality federated scheduling has a capacity augmentation bound of $2 + 2\sqrt{2}$ and $[(5 + \sqrt{5})/2]$ for dual- and multi-criticality systems, respectively. Moreover, they also derived a capacity augmentation bound of $((11m/3m - 3))$ two dual-criticality systems with both high- and low-utilization tasks.

3) **Global Scheduling:** For multiple DAGs with implicit deadlines, Li et al. [14] proved a bound of $4 - (2/m)$ under GEDF, this bound is further improved to $[(3 + \sqrt{5})/2]$, which is proved to be tight when the number $m$ of cores is sufficiently large. Moreover, Li et al. [18] proved a bound of $2 + \sqrt{3}$ under global rate monotonic scheduling without decomposition.

Moreover, for a single recurrent DAG with arbitrary deadline scheduled by GEDF, Baruah et al. [12] proved a bound of 2.5. In summary, prior work on capacity augmentation bounds is either restricted to a single recurrent DAG task or restricted to a set of multiple DAG tasks with implicit deadlines.

**B. Response Time Analysis**


Most RTA-based methods for multi-DAGs cannot provide guaranteed augmentation bounds. Moreover, unlike the capacity bound analysis that can provide a simple linear time schedulability test requiring no knowledge about DAG’s internal structure, RTA-based schedulability tests suffer from the complexity intrinsic in computation, which often have a (pseudo-)polynomial time complexity, and they require to explore DAG’s internal structure.

### III. Model

We consider a sporadic task set $\tau$ that consists of $n$ tasks $\tau = \{\tau_1, \ldots, \tau_n\}$. Each task $\tau_k$ is associated with a period $P_k$ and a relative deadline $D_k$, and its execution has a DAG structure. The $x$th subtask of task $\tau_k$ is represented by vertex $v^x_k$ in the DAG. If there is a directed edge from vertex $v^x_k$ to vertex $v^{y}_k$, then $v^x_k$ is $v^y_k$’s predecessor. A subtask cannot start its execution until the completion of all its predecessors. Each vertex $v^x_k$ has its own worst-case execution time $C^x_k$.

We assume the tasks have constrained deadlines, i.e., each task’s relative deadline is no larger than its period, i.e., $\forall k, D_k \leq P_k$. We do not restrict our research on any DAG of particular types. More specifically, multiple source vertices and sink vertices are allowed, and the DAG is not necessary to be fully connected. Fig. 2 gives an example task that contains six subtasks in the DAG-structure.

---

**TABLE I**

<table>
<thead>
<tr>
<th>DAG tasks</th>
<th>resource</th>
<th>capacity</th>
<th>resource</th>
<th>capacity</th>
<th>resource</th>
<th>capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit deadline</td>
<td>2 for GEDF [13], [14], 3 for GDM [13]</td>
<td>$\frac{2 + \sqrt{5}}{2}$ for GEDF, $\frac{2 + \sqrt{3}}{2}$ for GDM [18]</td>
<td>2 [18]</td>
<td>$[2,4]$ [20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constrained deadline</td>
<td>$\beta + 2\sqrt{\beta + 1}$ for GEDF (this work)</td>
<td>$\emptyset$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arbitrary deadline</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

---

2 An optimal federated scheduling may not be a good scheduling strategy compared with an optimal scheduling algorithm.
Fig. 2. Example DAG task $\tau_k$ with volume $C_k = 11$ and critical path length $L_k = 8$.

We now introduce some useful notations related to a DAG task.

1) **Volume:** The sum of the worst-case execution time of all subtasks of $\tau_k$ is the volume of $\tau_k$

$$C_k = \sum_x C^x_k.$$  

Moreover, we denote by $C_k^\Sigma$ the total volume of the whole task system: $C_k^\Sigma = \sum_k C_k$.

2) **Utilization:** We define the utilization $u_k$ of a task $\tau_k$ as

$$u_k = \frac{C_k}{P_k}.$$  

Moreover, the total utilization of the task system is denoted as $U_k = \sum_k u_k$.

3) We define the maximum ratio of task period to deadline as

$$\beta = \max_k \frac{P_k}{D_k}.$$  

4) **Critical Path:** We use the critical path of $\tau_k$ as the longest path in $\tau_k$’s DAG (the length of a path is the total amount of the worst-case execution time associated with the vertices along that path). Let $L_k$ be the critical path length, and obviously, $L_k \leq C_k$.

For example, in Fig. 2, the volume of $\tau_k$ is $C_k = 11$, and the utilization of $\tau_k$ is $u_k = 11/9$. The critical path (marking in red) starts from vertex $v^1_2$, goes through $v^3_4$ and ends at vertex $v^6_3$, so the critical path length of the DAG task $\tau_k$ is $L_k = 1 + 2 + 5 = 8$.

A task $\tau_k$ releases an infinite number of jobs recurrently, and the time interval between the release time of any two adjacent jobs is no less than period $P_k$. All of the jobs released by the same task have the same DAG-structure. In particular, the volumes and the critical path lengths of all jobs generated by a task $\tau_k$ are the same as those of task $\tau_k$.

Without loss of generality, $J_{k,a}$ denotes the $a$th job instance of task $\tau_k$, and the $x$th vertex of $J_{k,a}$ is represented as $v^x_{k,a}$. Let $r_{k,a}$ and $d_{k,a}$ be the absolute release time and absolute deadline of job $J_{k,a}$, respectively. All the vertices of $J_{k,a}$ are required to be executed after its release time $r_{k,a}$ and the execution must be completed on or before its deadline $d_{k,a}$. The interval $[r_{k,a}, d_{k,a}]$ is also known as the scheduling window of the job $J_{k,a}$, with a length of $D_k = d_{k,a} - r_{k,a}$ [as demonstrated in Fig. 3].

Moreover, we say that a job is unfinished if the job has been released but not completed yet. Any unfinished job must contain some vertices (subjobs) that are unfinished. To carry the analysis, here we define the notion of remaining volume and remaining critical path length for an unfinished job.

1) **Remaining Volume:** The remaining volume equals the total volume minus the part of its volume that has already been executed.

2) **Remaining Critical Path Length:** The remaining critical path length is total unfinished workload of the vertices in the longest path of the DAG.

For example, in the example DAG task shown in Fig. 2, if $v^1_2$ and $v^2_4$ are completely executed, and $v^8_3$ is partially executed for 1 time unit (out of 2), the remaining volume is $1+1+1+5 = 8$, and the remaining critical path length is $1 + 5 = 6$.

A. Runtime Scheduling and Schedulability

The task set is scheduled by GEDF scheduling algorithm on $m$ identical unit-speed processing cores. Under GEDF, at each time instant the scheduler selects the highest-priority ready vertices (at most $m$) for execution. Vertices of the same task share the same priority (ties are broken arbitrarily) and a vertex of a task with an earlier absolute deadline has a higher priority than a vertex of a task with a later absolute deadline. In particular, vertex-level preemption and migration are both permitted in GEDF. Without loss of generality, we assume the task system starts at time 0 (i.e., the first job of the system is released at time 0). The task set is schedulable if all jobs released by all tasks in $\tau$ meet their deadlines.

**Lemma 1** (Necessary Conditions for Schedulability [14]):

A task set $\tau$ is not schedulable (by any scheduler) unless the following conditions hold:

1) The critical path length of each task $\tau_k$ is less than its deadline, i.e.,

$$\forall k : L_k \leq D_k.$$  

2) The total utilization $U_k^\Sigma$ is smaller than the number of cores, i.e.,

$$U_k^\Sigma \leq m.$$  

Clearly, if (2) is violated for some task, then its deadline is doomed to be violated in the worst case, even if it is executed...
More precisely, for each job $J_k$, we assume that all the other jobs can meet their deadlines. Another job $J_{j,b}$ of $\tau_j$ can interfere with $J_{k,a}$ if the following conditions hold:

1) At some time point, $J_{j,b}$ and $J_{k,a}$ are both unfinished (this implies the scheduling windows of $J_{j,b}$ and $J_{k,a}$ are overlapped, assuming that $J_{j,b}$ meets its deadline).

2) The absolute deadline of $J_{j,b}$ is no later than the absolute deadline of $J_{k,a}$, i.e., $d_{j,b} \leq d_{k,a}$.

For any task $\tau_j$ we distinguish its jobs that may interfere with $J_{k,a}$ into two types by considering whether their scheduling windows are fully contained in the scheduling window of $J_{k,a}$ (see in Fig. 4).

1) Carry-in Jobs: A carry-in job ($J_{j,b}$) must be released before the job of interest ($J_{k,a}$) and has an absolute deadline earlier than the absolute deadline of $J_{k,a}$, i.e., $r_{j,b} < r_{k,a} \land d_{j,b} \leq d_{k,a}$ [as shown in Fig. 4(a)].

2) Fall-in Jobs: A fall-in job’s ($J_{j,b}$) scheduling window is fully contained in the scheduling window of the job of interest ($J_{k,a}$). More specifically, $J_{j,b}$ is released after the release time of $J_{k,a}$ and the absolute deadline of $J_{j,b}$ is earlier than the absolute deadline of $J_{k,a}$, i.e., $r_{j,b} \geq r_{k,a} \land d_{j,b} \leq d_{k,a}$ [as shown in Fig. 4(b)].

Note that a job $J_{j,b}$ that is a carry-in job of $J_{k,a}$ does not interfere with $J_{k,a}$, if $J_{j,b}$ has finished before the release time $r_{k,a}$ of $J_{k,a}$. If the carry-in job $J_{j,b}$ of $J_{k,a}$ is unfinished at $r_{k,a}$, then $J_{j,b}$ can interfere with $J_{k,a}$ and we call the work that is from the carry-in jobs of $J_{k,a}$ and interferes with $J_{k,a}$ as carry-in work.

Definition 2 (Carry-in Work): For a job $J_{k,a}$ under analysis, the carry-in work, denoted by $\chi^{k,a}$, is the total work from the carry-in jobs executed in the scheduling window of $J_{k,a}$.

According to Definition 2, the work from a carry-in job $J_{j,b}$ to $J_{k,a}$ contributes to the carry-in work of $J_{k,a}$ if it is executed during the interval $[r_{k,a}, d_{j,b}]$ (recall that when analyzing the schedulability of $J_{k,a}$ we assume $J_{j,b}$ can meet its deadline).

Similarly, a fall-in job may not interfere with $J_{k,a}$ unless $J_{k,a}$ is unfinished at the release time of $J_{j,b}$. If $J_{j,b}$ interferes with $J_{k,a}$, the amount of interfering work from $J_{j,b}$ is $C_j$, which is called fall-in work.

Definition 3 (Fall-in Work): For a job $J_{k,a}$ under analysis, its fall-in work $F^{k,a}$ is the total work from the fall-in jobs released before $J_{k,a}$ finishes its execution.

Note that the fall-in work $F^{k,a}$ of $J_{k,a}$ not only consists of the work from $J_{k,a}$’s fall-in jobs, but also contains the work from $J_{k,a}$ itself.

Let $n_{j,b}^{k,a}$ be the number of $J_{k,a}$’s fall-in jobs that are released from the task $\tau_j$ (see an example in Fig. 5). The total amount
of the fall-in work of \(J_{k,a}\) is upper bounded by
\[ F_{k,a} \leq \sum_i n_i^{k,a} C_i = \sum_i u_i n_i^{k,a} P_i. \]  
\[ (4) \]

**Definition 4 (Remaining Window Length):** Let \(J_{j,b}\) be a carry-in job from task \(\tau_j\) for the analyzed job \(J_{k,a}\), the remaining window length of \(\tau_j\) is defined as
\[ \alpha_{j}^{k,a} = d_{j,b} - r_{k,a}. \]

Obviously, \(\alpha_{j}^{k,a} \leq D_j\) [see Fig. 4(a)]. Moreover, as shown in Fig. 5, the following inequality holds:
\[ D_k \geq \alpha_{j}^{k,a} + P_j - D_j + \left( n_j^{k,a} - 1 \right) P_j + D_j \]
\[ = \alpha_{j}^{k,a} + n_j^{k,a} P_j. \]  
\[ (5) \]

B. Progress Under Work-Conserving Scheduling

The GEDF satisfies work-conserving property: cores will never be idle if there are ready vertices waiting for execution. The work-conserving property guarantees the system to make progress whenever there is ready workload to execute. The progress can be guaranteed differently for two types of intervals.

1) **Complete Interval:** At any time point in a complete interval, all cores are busy.

2) **Incomplete Interval:** At any time point in an incomplete interval, at least one core is idle.

In order to coincide with the analysis undertaken in the following sections, this section considers a more general case of scheduling on \(m\) cores with speed \(\rho\). The following lemmas are given in [14].

**Lemma 2:** On a processing platform of core speed \(\rho\), the remaining critical path length of each unfinished job reduces by \(\rho t\) after an incomplete interval of length \(t\) is elapsed.

**Lemma 3:** On a processing platform of core speed \(\rho\), the total work in a time interval of length \(t\), in which the accumulated length of incomplete intervals is \(t^\ast\), is at least \(\rho(m - 1)t^\ast\).

By Lemmas 2 and 3, we can obtain the following lemma.

**Lemma 4:** For any interval \(I\) that falls in the scheduling window of job \(J_{k,a}\), i.e., \(I \subseteq [r_{k,a}, t_{k,a}]\), if \(J_{k,a}\) finishes after \(I\), then the total amount of work done during \(I\) is at least \(\rho m |I| - (m - 1)L_k\), where \(L_k\) is the critical path length of \(\tau_k\).

**Proof:** We first prove that the accumulated length of incomplete intervals in \(I\), denoted by \(x\), is no more than \(L_k/\rho\). We prove this by contradiction, assuming \(x > L_k/\rho\). According to Lemma 2, \(J_{k,a}\)’s critical path length reduces by \(\rho \cdot x\) after all the incomplete intervals with the total length \(x\) are elapsed. Therefore, we can conclude that the critical path length reduces by more than \(L_k\) at the end of \(I\), which leads to a contradiction as the length of the critical path is at most \(L_k\).

By now, we know that the accumulated length of the incomplete intervals in \(I\) is at most \(L_k/\rho\). By Lemma 3, the total amount of work done during \(I\) is at least \(\rho m |I| - (m - 1)L_k\).

**Lemma 4** implies a lower bound of the amount of workload that must be done during an interval when some jobs are unfinished. This lemma will be used in the proofs of Section V-B.

V. ANALYSIS

This section presents our schedulability analysis and the capacity augmentation bound.

The main idea of our analysis is as follows. For any given positive number \(\epsilon\), we formulate a speed function \(\rho(\epsilon)\), and assume that the task set is run on \(m\) cores with speed up \(\rho(\epsilon)\). Then, for every job released from the task system, we can use a function of \(\epsilon\) to bound its carry-in work. For every job, the bounded carry-in work leads to bounded interference from other tasks, and hence GEDF can successfully schedule all of them. The infimum of the speed function \(\rho(\epsilon)\) eventually implies the capacity augmentation bound. In the following, Section V-A derives an upper bound for carry-in work, based on which, the proof for a capacity augmentation bound is presented in Section V-B.

A. Upper Bound for Carry-in Work

In the following, we show that the carry-in work for a job under analysis can be well bounded if scheduled on \(m\) \(\rho\)-speed cores. First, for the cores with speed \(\rho \geq 1\), a straightforward bound for carry-in work of the analyzed job \(J_{k,a}\) is as follows.

**Lemma 5:** If the core speed \(\rho \geq 1\), the carry-in work \(\chi_{k,a}\) for job \(J_{k,a}\) is bounded by
\[ \chi_{k,a} \leq \beta \sum_i u_i D_i. \]  
\[ (6) \]

**Proof:** Using \(J_1\) to denote the set of carry-in jobs of \(J_{k,a}\) that are unfinished at time \(r_{k,a}\), then we have
\[ \chi_{k,a} \leq \sum_{j,b \in J_1} u_j P_j \leq \beta \sum_{j,b \in J_1} u_j D_j \leq \beta \sum_i u_i D_i. \]
The last step of the above inequality is because that each constrained-deadline task \( \tau_i \) has at most one job to be the carry-in job of \( J_{k,a} \). This completes the proof.

For the cores with speed \( \rho \) strictly larger than 1, by representing the infimum of core speed \( \rho \) as a function, the carry-in-work bound for the analyzed job \( J_{k,a} \) can be further refined as shown in Lemma 6, and this is one of the basic result of this paper.

Lemma 6: If the core speed \( \rho \geq \rho(\epsilon) \) (where \( \epsilon > 0 \)), the carry-in work \( \chi^{k,a} \) for job \( J_{k,a} \) is bounded by
\[
\chi^{k,a} \leq \beta (1 + \epsilon) \sum_i u_i \alpha_i^{k,a}
\]
(7)

where
\[
\rho(\epsilon) = \beta (1 + \epsilon) + \left( 1 + \frac{1}{\epsilon} \right) \left( 1 - \frac{1}{m} \right).
\]
(8)

(Recall that \( \alpha_i^{k,a} \) is the remaining window length of task \( \tau_i \) as defined in Definition 4.)

Proof: We prove the lemma by an induction to jobs in the order of their release time. The job of interest is denoted as “\( J_{k,a} \)” at each induction step.

Base Case: If \( J_{k,a} \) is the first job released in the system, i.e., released at time 0, no carry-in jobs are released before \( J_{k,a} \), implying that \( \chi^{k,a} = 0 \), and \( \alpha_i^{k,a} = 0 \) for each \( \tau_i \in \tau \).

Therefore, the condition (7) trivially holds
\[
\chi^{k,a} = 0 \leq \beta (1 + \epsilon) \sum_i u_i \alpha_i^{k,a} = 0.
\]

Inductive Step: For the case that \( J_{k,a} \) is not the first job released in the system, we have the inductive hypothesis: every job \( J_{i,b} \) released earlier than \( J_{k,a} \) satisfies
\[
\chi^{i,b} \leq \beta (1 + \epsilon) \sum_i u_i \alpha_i^{i,b}.
\]
(9)

In the following we prove that (7) holds for job \( J_{k,a} \). First, the condition (7) trivially holds if \( \alpha_i^{k,a} > [D_j]/(1 + \epsilon) \), for every carry-in job \( J_{j,b} \) of \( J_{k,a} \). The reason is as follows. From Lemma 5, we have
\[
\chi^{k,a} \leq \beta \sum_j u_j D_j
\]
\[
< \beta (1 + \epsilon) \sum_j u_j \alpha_j^{k,a} \left[ \alpha_j^{k,a} > \frac{D_j}{1 + \epsilon} \right].
\]

Therefore, in the following we only consider the case such that at least one unfinished carry-in job \( J_{j,b} \) satisfies \( \alpha_j^{k,a} \leq [D_j]/(1 + \epsilon) \). Then by \( D_j = r_{k,a} - r_{j,b} + \alpha_j^{k,a} \) and letting \( \Delta = r_{k,a} - r_{j,b} \), we have
\[
\Delta \geq \frac{\epsilon}{1 + \epsilon} D_j.
\]
(10)

On the other hand, we have (see Fig. 6 for intuition)
\[
\Delta \geq \alpha_i^{i,b} + \sum_i u_j P_i + \epsilon n_i^{i} P_i + D_i - \alpha_i^{k,a}
\]
\[
\geq \alpha_i^{i,b} + \sum_i u_j P_i + \epsilon n_i^{i} P_i - \alpha_i^{k,a}
\]
(11)

where \( n_i^{i} \) denotes the number of jobs that are released after the release time \( r_{j,b} \) of \( J_{j,b} \) and whose next job is released before the release time \( r_{k,a} \) of \( J_{k,a} \).

Note that \( J_{j,b} \) has not finished at time \( r_{k,a} \). According to Lemma 4, the total amount of work done during \( [r_{j,b}, r_{k,a}] \), denoted by \( W^A \), is at least
\[
W^A \geq \rho m \Delta - (m - 1) L_j.
\]
(12)

The work of \( W^A \) comes from three sets of jobs.

1. \( J_A \): the set of carry-in jobs of \( J_{j,b} \).
2. \( J_B \): the set of carry-in jobs of \( J_{k,a} \).
3. \( J_C \): the set of jobs that entirely fall in \( [r_{j,b}, r_{k,a}] \).

For example, in Fig. 6, \( J_A = \{ J_{1,c}, J_{1,d} \} \) (in red rectangles), \( J_B = \{ J_{1,c+2}, J_{1,d} \} \) (in blue rectangles) and \( J_C = \{ J_{1,c+1} \} \) (in green rectangles). Obviously, \( J_A \cup J_B \cap J_C = \emptyset \), and in general \( J_A \cap J_B \neq \emptyset \).

Let \( J_A' = J_A - J_B \). We use \( W_A \) to denote the total amount of work done by jobs in \( J_A' \) (for \( x = A', A, B, C \)), the total amount of work \( W^A \) done during \( [r_{j,b}, r_{k,a}] \) can be divided into three parts
\[
W^A = W_A + W_B + W_C.
\]
(13)

In the following, we derive an upper bound for each part above, respectively.

Upper Bound of \( W_A \): Since the work in \( W_A \) is executed in the interval between the release time \( r_{j,b} \) of \( J_{j,b} \) and the absolute deadline \( d_{j,b} \) of \( J_{j,b} \), \( W_A' \) is included in the carry-in work \( \chi^{i,b} \) of \( J_{j,b} \), i.e., \( W_A' \leq \chi^{i,b} \), and by the inductive hypothesis (9), we have
\[
W_A' \leq \beta (1 + \epsilon) \sum_i u_i \alpha_i^{i,b}.
\]
(14)

Upper Bound of \( W_B \): We observe that the total amount of work by the carry-in jobs of \( J_{k,a} \), denoted by \( C_j^{k,a} \), can be divided into two parts.

1. The work done before or at the release time \( r_{k,a} \) of \( J_{k,a} \).
   This part includes \( W_B \).
2. The work done after the time \( r_{k,a} \), which equals \( \chi^{k,a} \).

Therefore, we have
\[
C_j^{k,a} \geq W_B + \chi^{k,a}.
\]
(15)

Each constrained-deadline task \( \tau_i \) has at most one job to be the carry-in job of \( J_{k,a} \). Thus, the total amount of work \( C_j^{k,a} \) from the carry-in jobs of \( J_{k,a} \) has an upper bound \( C_j^{k,a} \leq \sum_i u_i P_i \), and combining this with (15) yields
\[
W_B \leq \sum_i u_i P_i - \chi^{k,a}.
\]
(16)

Upper Bound of \( W_C \): For each \( \tau_i \in \tau \), recall that \( n_i^{\Delta} \) is the number of jobs that are released after the release time \( r_{j,b} \) of \( J_{j,b} \) and whose next job is released before the release time \( r_{k,a} \) of \( J_{k,a} \) [defined right after (11)]. The total amount of work \( W_C \) from \( J_C \) can be calculated as
\[
W_C = \sum_i u_i n_i^{\Delta} P_i.
\]
(17)
\[ \Delta = r_{k,a} - r_{j,b} \geq \alpha_i^{j,b} + n_i^P_i + P_i - \chi_i^{k,a} \]

and since by (12), we have
\[ \sum_{i} u_i \alpha_i^{j,b} + \sum_{i} u_i n_i^P_i + \sum_{i} u_i P_i - \chi_i^{k,a} \]

by which we finally get
\[ \Delta \geq r_{k,a} - r_{j,b} \geq \alpha_i^{j,b} + n_i^P_i + P_i - \chi_i^{k,a} \]

and by (12), we have
\[ \chi_i^{k,a} \leq \beta(1 + \epsilon) \sum_{i} u_i ^{j,b} + n_i^P_i + \sum_{i} u_i \leq \chi_i^{k,a} \]

and since \( \sum_{i} u_i \leq m \) and \( L_j \leq D_j \), we have
\[ \chi_i^{k,a} \leq \beta(1 + \epsilon) \left( m \Delta + \sum_{i} u_i \alpha_i^{j,b} \right) - \rho m \Delta + (m - 1)D_j \]

and by \( \Delta \geq (\epsilon/[1 + \epsilon])D_j \), we have
\[ \chi_i^{k,a} \leq \beta(1 + \epsilon) \left( m \Delta + \sum_{i} u_i \alpha_i^{j,b} \right) - \rho m \Delta + (m - 1)D_j \]

and since \( \rho \geq \beta(1 + \epsilon) + (\epsilon/[1 + \epsilon])(1 - (1/m)) \), we have
\[ \chi_i^{k,a} \leq \left( 1 + \epsilon \right)(m \Delta + \left( \epsilon + \frac{1}{\epsilon} \right)(m - 1) \Delta \]

by which we finally get \( \chi_i^{k,a} \leq \beta(1 + \epsilon) \sum_{i} u_i \alpha_i^{j,b} \).

**B. Upper Capacity Augmentation Bound**

In this section, we propose an capacity augmentation bound for the DAG tasks with constrained deadlines.

Recall that we can bound the fall-in work \( F_i^{k,a} \) by (4), and Lemma 6 bounds the carry-in work \( \chi_i^{k,a} \), so by now we have bounded the total amount of work to be executed in the scheduling window of \( J_{k,a} \), the job under analysis. Next, we will present a lemma that identifies core speeds for the platform to be able to finish this total amount of work in the scheduling window of \( J_{k,a} \), and thus guarantee the schedulability.

**Lemma 7**: A task set that satisfies the necessary conditions in Lemma 1 is schedulable under GEDF on a multicore platform with core speed \( \rho \geq \beta(1 + \epsilon) + (\epsilon/[1 + \epsilon])(1 - (1/m)) \) (where \( \epsilon > 0 \)), i.e., GEDF has a capacity augmentation bound of \( \beta(1 + \epsilon) + (\epsilon/[1 + \epsilon])(1 - (1/m)) \), where \( \beta = \max((P_i/D_i)) \).

Proof: We prove this theorem by contradiction. Suppose an arbitrary job \( J_{k,a} \) misses its deadline. It implies that all the work done during the scheduling window \( [r_{k,a}, d_{k,a}] \) of \( J_{k,a} \) (the length of which is \( D_k \)) can interfere with \( J_{k,a} \) (including \( J_{k,a} \)’s work).

We use \( W \) to denote the total amount of work that has been done in \( [r_{k,a}, d_{k,a}] \). Since \( J_{k,a} \) misses deadline, we know
\[ W \leq \chi_i^{k,a} + F_i^{k,a} \]

Since \( J_{k,a} \) has not finished at its absolute deadline \( d_{k,a} \), by Lemma 4, we have
\[ W \geq \rho m D_k - (m - 1)L_k \]
\[ \geq (1 + (\rho - 1)m)D_k \]
\[ \geq (\rho m D_k - (m - 1)L_k) \]
\[ \geq (1 + (\rho - 1)m)D_k \]

Then by (18) and (19), as well as the upper bounds for \( \chi_i^{k,a} \) in Lemma 6 and for \( F_i^{k,a} \) in (4), we have
\[ (1 + (\rho - 1)m)D_k \leq \beta(1 + \epsilon) \sum_{i} u_i \alpha_i^{k,a} + \sum_{i} u_i n_i^{k,a} \]
\[ \Rightarrow (1 + (\rho - 1)m)D_k \leq \beta(1 + \epsilon) \sum_{i} u_i \left( \alpha_i^{k,a} + n_i^{k,a} \right) P_i \]
\[ \Rightarrow (1 + (\rho - 1)m)D_k \leq \beta(1 + \epsilon) \sum_{i} u_i D_k \] [from (5)]
\[ \Rightarrow (1 + (\rho - 1)m)D_k \leq \beta(1 + \epsilon) \sum_{i} u_i \leq m \]
\[ \Rightarrow 1 + (\rho - 1)m \leq \beta(1 + \epsilon) m \]
\[ \Leftrightarrow \rho \leq \beta(1+\epsilon) + 1 - \frac{1}{m} \]
\[ \Rightarrow \rho < \beta(1+\epsilon) + \left(\epsilon + \frac{1}{\epsilon}\right) \left(1 - \frac{1}{m}\right) \]
\[ \therefore m > 1, \epsilon + \frac{1}{\epsilon} \geq 2 \].

It contradicts to the precondition \( \rho \geq \beta(1+\epsilon)+(\epsilon+(1/\epsilon))(1-(1/m)) \), so assumption is not true and the lemma is proved. ■

Note that the capacity augmentation bound in Lemma 7 contains an open variable \( \epsilon \). Lemma 7 holds for any \( \epsilon > 0 \), and our target is to achieve a bound as low as possible. The following lemma gives the value of \( \epsilon \) to make the bound \( \beta(1+\epsilon) + (\epsilon + (1/\epsilon))(1-(1/m)) \) to reach its minimum.

**Lemma 8:** \( \beta(1+\epsilon) + (\epsilon + (1/\epsilon))(1-(1/m)) \) reaches its minimum \( \beta + 2\sqrt{(\beta + 1 - 1/(1/m))(1-(1/m))} \) with \( \epsilon = \sqrt{[(1-(1/m))/(\beta + 1 - 1/(1/m))]}. \)

*Proof:* We rewrite the bound \( \beta(1+\epsilon) + (\epsilon + (1/\epsilon))(1-(1/m)) \) as

\[ \beta(1+\epsilon) + \left(\epsilon + \frac{1}{\epsilon}\right) \left(1 - \frac{1}{m}\right) = \beta + A + B \]

where \( A = (\beta + 1 - (1/m))\epsilon, \ B = (1 - (1/m))(1/\epsilon) \).

Since \( A + B \geq 2\sqrt{AB} \), we know the lower bound of \( \beta + A + B \)

\[ \beta + A + B \geq 2\sqrt{AB} = \beta + 2\sqrt{\beta + 1 - \frac{1}{m} \left(1 - \frac{1}{m}\right)} . \]

Since \( A + B \) reaches its minimum \( 2\sqrt{AB} \) with \( A = B \), we can solve the desired \( \epsilon \) with

\[ \left(\beta + 1 - \frac{1}{m}\right) \epsilon = \left(1 - \frac{1}{m}\right) \frac{1}{\epsilon} \]

by which we get \( \epsilon = \sqrt{[(1-(1/m))/(\beta + 1 - 1/(1/m))]}. \) ■

Now, by substituting the bound in Lemma 7 by its minimum we can conclude the main result of this paper.

**Theorem 1:** A task set that satisfies the necessary conditions in Lemma 1 is schedulable under GEDF on a multicore platform with core speed \( \rho \geq \beta + 2\sqrt{(\beta + 1 - 1/(1/m))(1-(1/m))} \), i.e., GEDF has a capacity augmentation bound of \( \beta + 2\sqrt{(\beta + 1 - 1/(1/m))(1-(1/m))} \), where \( \beta = \max_i\{P_i/D_i\} \).

We can state Theorem 1 in the form of a direct schedulability test on unit-speed cores.

**Corollary 1:** On \( m \) unit-speed cores, where \( m > 1 \), if a sporadic task set \( \tau \) with constrained deadlines satisfies the following two conditions:

\[ U_\Sigma \leq \frac{m}{\beta + 2\sqrt{\left(\beta + 1 - \frac{1}{m}\right)(1 - \frac{1}{m})}} \]

\[ \forall k : L_k \leq \frac{D_k}{\beta + 2\sqrt{\left(\beta + 1 - \frac{1}{m}\right)(1 - \frac{1}{m})}} \]

where \( \beta = \max_i\{P_i/D_i\} \), then \( \tau \) is schedulable by GEDF.

**C. Lower Capacity Augmentation Bound**

This section gives an example to show the lower bound of the capacity augmentation bound.

The example is constructed as shown in Fig. 7. The task set contains two tasks. One task \( \tau_1 \) is structured as a single vertex with workload \( x \) followed by \( nm \) vertices with workload \( y \). Its critical path length \( L_1 = x + y \) and so is its deadline. The period of \( \tau_1 \) is set to be \( \beta(x+y) \), and moreover, the utilization \( u_1 \) is set to be \( m - 1 \)

\[ m - 1 = \frac{x + nmy}{\beta(x+y)} \] (20)

The other task \( \tau_2 \) has a single vertex with workload, deadline, and period equal to \( x + y - \frac{x}{\beta} \), and thus the critical path length \( L_2 \) of \( \tau_2 \) is \( x + y - \frac{x}{\beta} \) and the utilization \( u_2 \) of \( \tau_2 \) is \( 1 \).

Obviously, the necessity conditions (2) and (3) hold: \( U_\Sigma = u_1 + u_2 \leq m, L_1 \leq D_1 \) and \( L_2 \leq D_2 \). During the execution, \( \tau_1 \) is released at the absolute time 0, and \( \tau_2 \) is released at time \( \frac{x}{\beta} + 1 \). The execution is shown in Fig. 8.

We want to generate an example, so we want \( \tau_2 \) to miss its deadline. In order for this to occur, we must have

\[ x + y - \frac{x}{\rho} + 1 < \frac{ny + x + y - \frac{x}{\rho}}{\rho} . \] (21)

Reorganizing and combining (20) and inequality (21), we get

\[ \rho < \frac{(n + 1)m\beta + 2(nm - (m - 1)\beta)}{2(nm - (m - 1)\beta) + 2((m - 1)\beta - 1) + (\sqrt{(n + 1)^2\beta^2 + 4m((m - 1)\beta - 1)(nm - (m - 1)\beta)}}{2(nm - (m - 1)\beta) + 2((m - 1)\beta - 1)} . \] (22)

In (22), for large enough \( nm \), we have

\[ \rho < \frac{(\beta + 2)nm + \sqrt{\beta^2 + 4\beta}n^2m^2}{2nm} \]

\[ \Leftrightarrow \rho < \frac{\beta + \sqrt{\beta^2 + 4\beta} + 1}{2} . \] (23)

So there exists an example for any speed-up \( \rho \) that satisfies the above conditions. Therefore, the capacity augmentation bound...
required by GEDF is at least \([(\beta + \sqrt{\beta^2 + 4\beta})/2] + 1\). In particular, the bound is \([(3 + \sqrt{5})/2]\) for implicit deadline task sets.

**Corollary 2:** The gap ratio of the bound in Theorem 1 to the optimal one does not exceed 1.47.

**Proof:** By dividing the upper bound in Theorem 1 by the lower bound in (23) and for large \(m\), we obtain the upper bound of the ratio of the gap ratio under analysis as follows:

\[
\frac{2\beta + 4\sqrt{\beta + 1}}{\beta + \sqrt{\beta^2 + 4\beta} + 2}.
\]

The maximum value of (24) is 1.4641, when \(\beta \approx 2\).

**VI. EXPERIMENTS**

In this evaluation, we compare the schedulability tests based on Corollary 1 of this paper (denoted by CAP) and [13, Th. 21] (denoted by BON), both of which are linear-time schedulability test conditions for constrained-deadline DAG tasks under GEDF.

The task sets are generated using the Erdös–Rényi method \(G(n_k, p)\) [33]. For each task \(t_k\), the number of vertices is randomly chosen in the range \([50, 250]\) and the worst-case execution time of each vertex is randomly picked in the range \([50, 100]\). A valid period \(P_k\) is generated according to its target utilization, and the deadline \(D_k\) is uniformly chosen in \([P_k/\beta, P_k]\). For each possible edge we generate a random value in the range \([0, 1]\) and add the edge to the graph only if the generated value is less than a predefined threshold \(p\). In general the critical path of a DAG generated using the Erdös–Rényi method becomes longer as \(p\) increases, which makes the task more sequential. We use \(n\) to denote the number of tasks in a task set and \(m\) the number of cores. For each parameter configuration, we randomly generate 10,000 task sets. We compare the acceptance ratio of CAP and BON. The acceptance ratio is the ratio between the number of task sets deemed to be schedulable by a method and the total number of task sets that participate in the experiment (with a specific parameter configuration).

Fig. 9 reports the acceptance ratio of the tests as a function of the total utilization \(U_\Sigma\), where we set \(n = 20, m = 16, \beta = 2, p = 0.25\). We observe that CAP method clearly outperforms the BON method.

Fig. 10 shows the results with different number of cores, with a fixed utilization \(U_\Sigma = 4\), and set \(n = 20, \beta = 2, p = 0.25\). Since the total volume is fixed now, it becomes easier to successfully schedule a task set with more cores.

The experimental result shows that CAP requires less cores than BON to make the task set to be schedulable.

Fig. 11 shows the results with different \(p\) (which determines the intratask parallelism of tasks), with \(U_\Sigma = 4, n = 20, m = 16, \beta = 2.5\). We observe that CAP, the schedulability is better for tasks with higher parallelism. This is because, for a task with fixed volume, a more parallel structure in general leads to a shorter critical path, and thus more laxity, which is beneficial to schedulability. However, this trend is very weak for BON. Fig. 11 shows that BON has a low acceptance ratio ranging from 0.2 to 0.3 with different parallelism degrees, which clearly implies the superiority of CAP over BON in exploring the laxity of the tasks.

Fig. 12 shows the results with different \(\beta\) (which determines the relative deadlines of tasks), with \(U_\Sigma = 2, n = 20, m = 16,\) and \(p = 0.25\). For both tests, the schedulability ratio decreases when \(\beta\) increases. However, CAP can tolerate the increase of \(\beta\) much better than BON.

**VII. CONCLUSION**

In this paper, we consider multiple parallel tasks in the DAG model, and prove that for parallel tasks with constrained deadlines the capacity augmentation bound of GEDF is \(\beta + 2\sqrt{\beta + 1 - (1/m))(1 + (1/m))}\), where \(\beta = \max_i(P_i/D_i)\).
This is the first capacity augmentation bound for DAG tasks with constrained deadlines. Compared with existing schedulability test for the same problem setting also with linear-time complexity, the capacity augmentation result reported here performs better in terms of acceptance ratio. Moreover, we prove that the optimal capacity augmentation bound cannot be lower than \((\beta + 2 + \sqrt{\beta^2 + 4\beta})/2\). The ratio of our bound to the optimal one does not exceed 1.47. As the future work, we will generalize the result of this paper to arbitrary-deadline tasks.

**REFERENCES**


Xu Jiang received the B.S. degree in computer science from Northwestern Polytechnical University, Xi’an, China, in 2009, the M.S. degree in computer architecture from the Graduate School of the Second Research Institute, China Aerospace Science and Industry Corporation, Beijing, China, in 2012, and the Ph.D. degree from the Laboratory of Embedded Systems, Beihang University, Beijing, in 2018. He is currently researching as a Research Assistant with Hong Kong Polytechnic University, Hong Kong. His current research interests include real-time systems, parallel and distributed systems, and embedded systems.

Shuangshuang Chang received the M.S. degree in computer technology from Northeastern University, Shenyang, China, in 2016, where she is currently pursuing the Ph.D. degree. Her current research interests include embedded real-time system, scheduling analysis in mixed-criticality system, and security mechanism of cyber-physical systems.

Zhishan Guo received the B.E. degree in computer science and technology from Tsinghua University, Beijing, China, in 2009, the M.Phil. degree in mechanical and automation engineering from the Chinese University of Hong Kong, Hong Kong, in 2011, and the Ph.D. degree in computer science from the University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, in 2016. He is an Assistant Professor with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, USA, and an Assistant Professor with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA. His current research interests include real-time scheduling, cyber-physical systems, and neural networks and their applications.

Qingxu Deng received the Ph.D. degree from Northeastern University, Shenyang, China, in 1997. He is currently a Full Professor with the School of Computer Science and Engineering, Northeastern University. His current research interests include multiprocessor real-time scheduling and formal methods in real-time system analysis.

Wang Yi (F’15) received the Ph.D. degree in computer science from the Chalmers University of Technology, Gothenburg, Sweden, in 1991. He is a Chair Professor with Uppsala University, Uppsala, Sweden. His current interests include models, algorithms, and software tools for building and analyzing computer systems in a systematic manner to ensure predictable behaviors. Dr. Yi was a recipient of the CAV 2013 Award for contributions to model checking of real-time systems, in particular the development of UPPAAL, the foremost tool suite for automated analysis and verification of real-time systems. For contributions to real-time systems, the Best Paper Awards of RTSS 2015, ECRTS 2015, DATE 2013, and RTSS 2009, the Outstanding Paper Award of ECRTS 2012, and the Best Tool Paper Award of ETAPS 2002. He is also on the steering committees of ACM EMSOFT (Co-Chair), ACM LCTES, and FORMATS. He serves frequently on technical program committees for a large number of conferences, and was the TPC Chair of TACAS 2001, FORMATS 2005, EMSOFT 2006, HSCC 2011, and LCTES 2012, and the Track/Topic Chair for RTSS 2008 and DATE from 2012 to 2014. He is a member of Academy of Europe (Section of Informatics).