A Capacity Augmentation Bound for Real-Time Constrained-Deadline Parallel Tasks Under GEDF

Jinghao Sun, Nan Guan, Xu Jiang, Shuangshuang Chang, Zhishan Guo, Qingxu Deng, and Wang Yi, *Fellow, IEEE*

***Abstract*—Capacity augmentation bound is a widely used quantitative metric in theoretical studies of schedulability analysis for directed acyclic graph (DAG) parallel real-time tasks, which not only quantifies the suboptimality of the scheduling algorithms, but also serves as a simple linear-time schedulability test. Earlier studies on capacity augmentation bounds of the sporadic DAG task model were either restricted to a single DAG task or a set of tasks with implicit deadlines. In this paper, we consider parallel tasks with constrained deadlines under global earliest deadline first policy. We first show that it is impossible to obtain a constant bound for our problem setting, and derive both lower and upper bounds of the capacity augmentation bound as a function with respect to the maximum ratio of task period to deadline. Our upper bound is at most 1.47 times larger than the optimal one. We conduct experiments to compare the acceptance ratio of our capacity augmentation bound with the existing schedulability test also having linear-time complexity. The results show that our capacity augmentation bound significantly outperforms the existing linear-time schedulability test under different parameter settings.**

***Index Terms*—Capacity augmentation bound, directed acyclic graph (DAG), global earliest deadline first (GEDF), parallel tasks, real-time scheduling, schedulability analysis.**

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. This work was supported in part by the Research Grants Council of Hong Kong under Grant ECS 25204216 and Grant GRF 15204917, in part by the University Grants Committee of Hong Kong under Project 1-ZVJ2 through The Hong Kong Polytechnic University, in part by the National Nature Science Foundation under Grant CNS 1755965, in part by the National Nature Science Foundation of China under Grant 61672140 and Grant 61472072, and in part by the Fundamental Research Funds for the Central Universities under Grant N172304025. This paper was recommended by Conference Chairperson B. Brandenburg. *(Corresponding author: Nan Guan.)*

J. Sun is with the School of Computer Science and Engineering, Northeastern University, Shenyang 110004, China, and also with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: jhsun@mail.dlut.edu.cn).

N. Guan and X. Jiang are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: nan.guan@polyu.edu.hk; jiangxu617@163.com).

S. Chang and Q. Deng are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110004, China (e-mail: changs1393587345@sina.co; dengqx@mail.neu.edu.cn).

Z. Guo is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32611 USA, and also with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO 65409 USA (e-mail: guozh@mst.edu).

W. Yi is with the School of Computer Science and Engineering, Northeastern University, Shenyang 110004, China, and also with the Department of Information Technology, Uppsala University, 752 36 Uppsala, Sweden (e-mail: yi@it.uu.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2857362

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 parallelisim, but also intratask parallelisim 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.

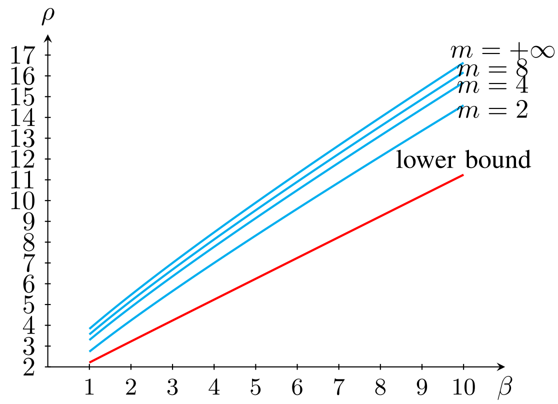


Fig. 1. Capacity bound as a function of β, and the red line represents the lower bound of capacity augmentation.

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 ρ if it can successfully schedule any task set τ on m cores of speed ρ as long as the compared scheduler can schedule τ 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 ρ if it can schedule any task set τ satisfying the following two conditions: a) the total utilization of τ is at most m/ρ 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 ρ, it is also guaranteed to have a resource augmentation bound of ρ. 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

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

where m is the number of processing cores and β 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 $(\beta + \sqrt{\beta^2 + 4\beta})/2 + 1$. Fig. 1 shows the figure of this capacity augmentation bound as a function of β.

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[1] 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 subcatagories: 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

[1]The test in [13] is for arbitrary-deadline DAG tasks, and thus also applicable to constrained-deadline DAG tasks considered in this paper.

TABLE I

STATE-OF-THE-ART RESOURCE AUGMENTATION BOUNDS (WITH RESPECT TO OPTIMAL SCHEDULING ALGORITHMS) AND CAPACITY AUGMENTATION BOUND FOR DAG TASKS (WHEN m IS INFINITELY LARGE)

DAG tasks	global scheduling		federated scheduling		decomposition-based	
	resource	**capacity**	**resource**	**capacity**	**resource**	**capacity**
implicit deadline	2 for GEDF [13], [14], 3 for GDM [13]	$\frac{3+\sqrt{5}}{2}$ for GEDF, $2+\sqrt{3}$ for GRM [18]	2 [18]		[2,4) [20]	
constrained deadline		$\beta + 2\sqrt{\beta + 1}$ **for GEDF (this work)**	$\emptyset$			
arbitrary deadline		$\emptyset$				

number of cores. With respect to any optimal federated scheduling algorithm,[2] 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 $(11m/[3m-3])$ for 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

For synchronous tasks with constrainted deadline, Chwa *et al.* [10] proposed an RTA-based analysis for GEDF scheduling algorithm, and Maia *et al.* [11] gave the anaylsis for GFP scheduling algorithm. Axer *et al.* [6] proposed an RTA-based analysis for fork-join tasks with arbitary deadline. Qamhieh *et al.* [15] gave an RTA-based analysis for GEDF scheduling of DAG-tasks with constrained deadline and a study of its sustainability. Parri *et al.* [32] proposed an RTA-based test for GEDF and GDM scheduling of DAG-tasks with arbitrary deadline. Melani *et al.* [21] proposed an RTA-based test for GEDF scheduling of conditional DAG-tasks with constrained deadline.

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 τ that consists of n tasks $\tau = \{\tau_1, \ldots, \tau_n\}$. Each task τ_k is associated with a *period* P_k and a *relative deadline* D_k, and its execution has a DAG structure. The xth subtask of task τ_k is represented by *vertex* v_k^x in the DAG. If there is a directed edge from vertex v_k^x to vertex v_k^y, then v_k^x is v_k^y's *predecessor*. A subtask cannot start its execution until the completion of all its predecessors. Each vertex v_k^x has its own *worst-case execution time* C_k^x.

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.

[2]An optimal federated scheduling may not be a good scheduling strategy compared with an optimal scheduling algorithm.

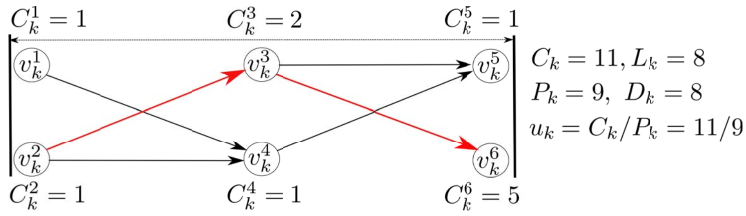


Fig. 2. Example DAG task τ_k with volume $C_k = 11$ and critical path length $L_k = 8$.

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 τ_k is the *volume* of τ_k

$$C_k = \sum_x C_k^x.$$

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

2) *Utilization:* We define the *utilization* u_k of a task τ_k as

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

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

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 τ_k as the longest path in τ_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, $L_k \le C_k$.

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

A task τ_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 τ_k are the same as those of task τ_k.

Without loss of generality, $J_{k,a}$ denotes the ath job instance of task τ_k, and the xth vertex of $J_{k,a}$ is represented as $v_{k,a}^x$. 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.

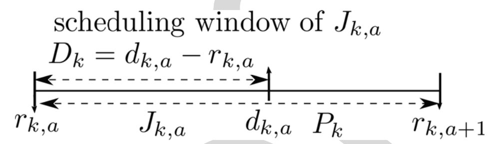


Fig. 3. Scheduling window $[r_{k,a}, d_{k,a}]$ of job $J_{k,a}$.

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_k^1 and v_k^2 are completely executed, and v_k^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 τ meet their deadlines.

Lemma 1 (Necessary Conditions for Schedulability [14]): A task set τ is not schedulable (by any scheduler) unless the following conditions hold.

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

$$\forall k : L_k \le D_k. \tag{2}$$

2) The total utilization $U_\sum$ is smaller than the number of cores, i.e.,

$$U_\sum \le m. \tag{3}$$

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

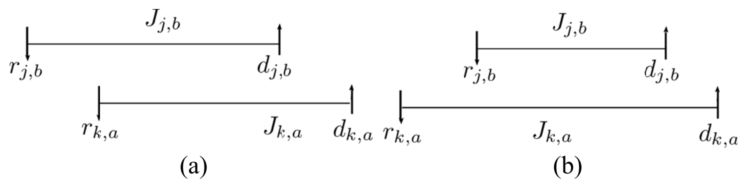


Fig. 4. Two types of jobs that may interfere with $J_{k,a}$. (a) $J_{j,b}$ is a carry-in job of $J_{k,a}$. (b) $J_{j,b}$ is a fall-in job of $J_{k,a}$.

exclusively on sufficiently many cores. If (3) is violated, then in the long term the worst-case workload of the system exceeds the processing capacity provided by the platform, and thus the backlog will increase infinitely which leads to deadline misses.

A scheduling algorithm S has a *capacity augmentation bound* ρ if any task set τ satisfying the following conditions is schedulable by S: 1) $\forall k : L_k \leq D_k/\rho$ and 2) $U_{\sum} \leq m/\rho$. The concept of *capacity augmentation bound* can be equivalently stated as follows [14] and [18]:

Definition 1 (Capacity Augmentation Bound for DAG Task System): A scheduling algorithm S has a *capacity augmentation bound* ρ if it can always schedule DAG task set τ on m cores of speed ρ as long as τ satisfies the above necessary conditions (2) and (3).

A scheduling algorithm with a smaller ρ is preferable and when $\rho = 1$ the scheduling algorithm S is optimal.

B. Overall Analysis Outline

The overall intuition behind the capacity bound analysis is to derive a sufficient condition, under which every released job can be successfully scheduled by GEDF on cores with speed ρ. More precisely, for each job $J_{k,a}$ under analysis, we derive a lower bound of the multicore resource that must be utilized to execute tasks in the scheduling window $[r_{k,a}, d_{k,a}]$ of $J_{k,a}$, and meanwhile, we derive an upper bound of the workload that must be executed by GEDF during the scheduling window $[r_{k,a}, d_{k,a}]$ of $J_{k,a}$. A sufficient condition for successfully scheduling tasks is that the resource's lower bound is larger than the workload's upper bound for all jobs. As we know that the lower resource bound increases with the core speed ρ and the upper workload bound decreases with the core speed ρ, we aim to find the minimum speed ρ to make the sufficient condition hold. Such a minimum speed ρ is the capacity augmentation bound as shown in Definition 1.

In the following, the upper workload bound is analyzed in Sections IV-A and V-A. Moreover, the lower resource bound is given in Section IV-B. Determining the infimum of speed ρ is given in Section V-B.

IV. Preliminary Results

In this section, we introduce some concepts and properties that will be useful in deriving the capacity augmentation bound in the next section.

A. Interference

Suppose we are analyzing the schedulability of an arbitrary job $J_{k,a}$, the ath instance of task τ_k, under GEDF scheduling. When analyzing $J_{k,a}$, we assume that all the other jobs can meet their deadlines. Another job $J_{j,b}$ of τ_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 τ_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.,$r_{j,b} < r_{k,a} \wedge d_{j,b} \leq d_{k,a}$ [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., $r_{j,b} \geq r_{k,a} \wedge d_{j,b} \leq d_{k,a}$ [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 $n_j^{k,a}$ be the number of $J_{k,a}$'s fall-in jobs that are released from the task τ_j (see an example in Fig. 5). The total amount

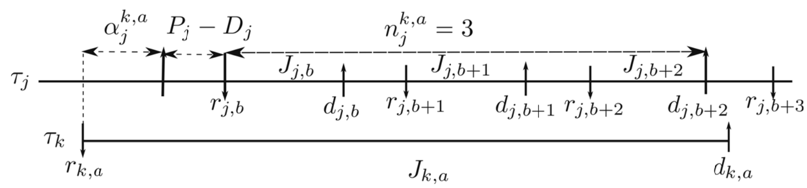


Fig. 5. Number of $J_{k,a}$'s fall-in jobs from τ_j is $n_j^{k,a} = 3$.

of the fall-in work of $J_{k,a}$ is upper bounded by

$$F^{k,a} \leq \sum_i n_i^{k,a} C_i = \sum_i u_i n_i^{k,a} P_i. \quad (4)$$

Definition 4 (Remaining Window Length): Let $J_{j,b}$ be a carry-in job from task τ_j for the analyzed job $J_{k,a}$, the remaining window length of τ_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:

$$\begin{aligned} D_k &\geq \alpha_j^{k,a} + P_j - D_j + \left(n_j^{k,a} - 1\right) P_j + D_j \\ &= \alpha_j^{k,a} + n_j^{k,a} P_j. \end{aligned} \quad (5)$$

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 ρ. The following lemmas are given in [14].

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

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

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

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

Proof: We first prove that the accumulated length of incomplete intervals in $\mathcal{I}$, denoted by x, is no more than L_k/ρ. 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 $\mathcal{I}$. 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 $\mathcal{I}$ is at most L_k/ρ. By Lemma 3, the total amount of work done during $\mathcal{I}$ is at least $\rho m|\mathcal{I}| - (m-1)L_k$. ■

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 ϵ, 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 ϵ 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 ρ-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

$$\chi^{k,a} \leq \beta \sum_i u_i D_i. \quad (6)$$

Proof: Using $\mathcal{J}_1$ to denote the set of carry-in jobs of $J_{k,a}$ that are unfinished at time $r_{k,a}$, then we have

$$\begin{aligned} \chi^{k,a} &\leq \sum_{J_{j,b} \in \mathcal{J}_1} u_j P_j \\ &\leq \beta \sum_{J_{j,b} \in \mathcal{J}_1} u_j D_j \quad \left[\because \beta = \max_i \left\{\frac{P_i}{D_i}\right\}\right] \\ &\leq \beta \sum_i u_i D_i. \end{aligned}$$

The last step of the above inequality is because that each constrained-deadline task τ_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 ρ strictly larger than 1, by representing the infimum of core speed ρ 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

$$\chi^{k,a} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a} \tag{7}$$

where

$$\rho(\epsilon) = \beta(1+\epsilon) + \left(\epsilon + \frac{1}{\epsilon}\right)\left(1 - \frac{1}{m}\right). \tag{8}$$

(Recall that $\alpha_i^{k,a}$ is the remaining window length of task τ_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 very 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 $J_{j,b}$ released earlier than $J_{k,a}$ satisfies

$$\chi^{j,b} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{j,b}. \tag{9}$$

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

$$\begin{aligned}\chi^{k,a} &\leq \beta\sum_j u_j D_j \\ &< \beta(1+\epsilon)\sum_j u_j \alpha_j^{k,a} \quad \left[\because \alpha_j^{k,a} > \frac{D_j}{1+\epsilon}\right].\end{aligned}$$

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

$$\Delta \geq \frac{\epsilon}{1+\epsilon} D_j. \tag{10}$$

On the other hand, we have (see Fig. 6 for intuition)

$$\begin{aligned}\Delta &\geq \alpha_i^{j,b} + P_i - D_i + n_i^{\Delta} P_i + D_i - \alpha_i^{k,a} \\ &\geq \alpha_i^{j,b} + n_i^{\Delta} P_i + P_i - \alpha_i^{k,a}\end{aligned} \tag{11}$$

where n_i^{Δ} 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^{Δ}, is at least

$$W^{\Delta} \geq \rho m \Delta - (m-1)L_j. \tag{12}$$

The work of W^{Δ} comes from three sets of jobs.

1) $\mathcal{J}_A$: the set of carry-in jobs of $J_{j,b}$.
2) $\mathcal{J}_B$: the set of carry-in jobs of $J_{k,a}$.
3) $\mathcal{J}_C$: the set of jobs that entirely fall in $[r_{j,b}, r_{k,a}]$.

For example, in Fig. 6, $\mathcal{J}_A = \{J_{i,c}, J_{l,d}\}$ (in red rectangles), $\mathcal{J}_B = \{J_{i,c+2}, J_{l,d}\}$ (in blue rectangles) and $\mathcal{J}_C = \{J_{i,c+1}\}$ (in green rectangles). Obviously, $(\mathcal{J}_A \cup \mathcal{J}_B) \cap \mathcal{J}_C = \emptyset$, and in general $\mathcal{J}_A \cap \mathcal{J}_B \neq \emptyset$.

Let $\mathcal{J}_A' = \mathcal{J}_A - \mathcal{J}_B$. We use W_x to denote the total amount of work done by jobs in $\mathcal{J}_x$ (for $x = A', A, B, C$), the total amount of work W^{Δ} done during $[r_{j,b}, r_{k,a}]$ can be divided into three parts

$$W^{\Delta} = W_{A'} + W_B + W_C. \tag{13}$$

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 $\chi^{j,b}$ of $J_{j,b}$, i.e., $W_{A'} \leq \chi^{j,b}$, and by the inductive hypothesis (9), we have

$$W_{A'} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{j,b}. \tag{14}$$

Upper Bound of W_B: We observe that the total amount of work by the carry-in jobs of $J_{k,a}$, denoted by $C^{k,a}$ 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

$$C^{k,a} \geq W_B + \chi^{k,a}. \tag{15}$$

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

$$W_B \leq \sum_i u_i P_i - \chi^{k,a}. \tag{16}$$

Upper Bound of W_C: For each $\tau_i \in \tau$, recall that n_i^{Δ} 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 $\mathcal{J}_C$ can be calculated as

$$W_C = \sum_i u_i n_i^{\Delta} P_i. \tag{17}$$

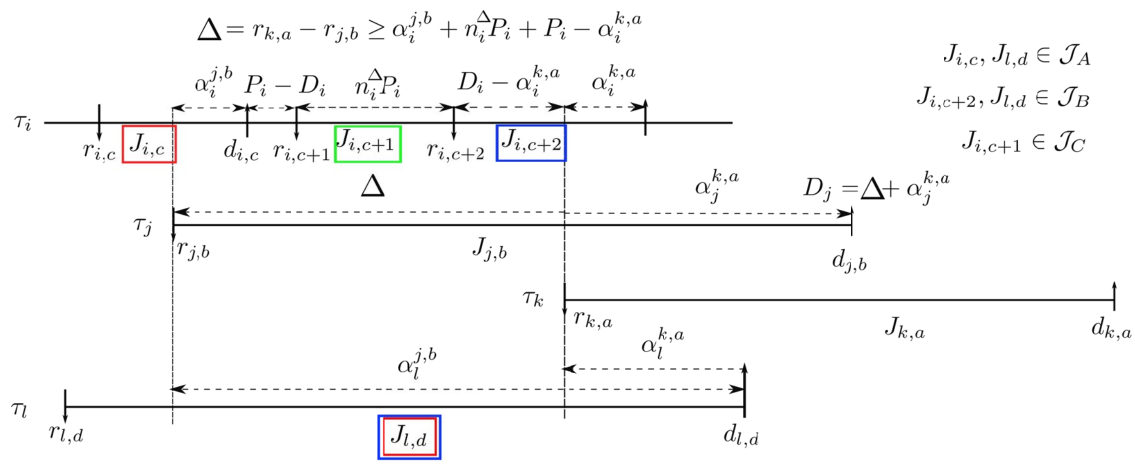


Fig. 6. Illustration for the proof of Lemma 6.

Putting (13), (14), (16), and (17) together, we have

$$W^{\Delta} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{j,b} + \sum_i u_i n_i^{\Delta} P_i + \sum_i u_i P_i - \chi^{k,a}$$
$$\leq \beta(1+\epsilon)\sum_i u_i \Big(\alpha_i^{j,b} + n_i^{\Delta} P_i + P_i\Big) - \chi^{k,a}$$
$$[\because \epsilon > 0, \beta > 1]$$

and by (12), we have

$$\chi^{k,a} \leq \beta(1+\epsilon)\sum_i u_i \Big(\alpha_i^{j,b} + n_i^{\Delta} P_i + P_i\Big)$$
$$- \rho m \Delta + (m-1) L_j$$
$$\leq \beta(1+\epsilon)\sum_i u_i \Big(\Delta + \alpha_i^{k,a}\Big) - \rho m \Delta$$
$$+ (m-1) L_j \quad [\because (11)]$$

and since $\sum_i u_i \leq m$ and $L_j \leq D_j$, we have

$$\chi^{k,a} \leq \beta(1+\epsilon)\left(m\Delta + \sum_i u_i \alpha_i^{k,a}\right) - \rho m \Delta + (m-1) D_j$$

and by $\Delta \geq (\epsilon/[1+\epsilon]) D_j$, we have

$$\chi^{k,a} \leq (\beta(1+\epsilon) - \rho) m \Delta + (m-1)\left(\epsilon + \frac{1}{\epsilon}\right)\Delta$$
$$+ \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a}$$

and since $\rho \geq \beta(1+\epsilon) + (\epsilon + (1/\epsilon))(1-(1/m))$, we have

$$\chi^{k,a} \leq \left(\epsilon + \frac{1}{\epsilon}\right)(1-m)\Delta + \left(\epsilon + \frac{1}{\epsilon}\right)(m-1)\Delta$$
$$+ \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a}$$

by which we finally get $\chi^{k,a} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a}$. ■

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 $F^{k,a}$ by (4), and Lemma 6 bounds the carry-in work $\chi^{k,a}$, 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 $\beta = \max_i\{(P_i/D_i)\}$.

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

$$W \leq \chi^{k,a} + F^{k,a}. \tag{18}$$

Since $J_{k,a}$ has not finished at its absolute deadline $d_{k,a}$, by Lemma 4, we have

$$W \geq \rho m D_k - (m-1) L_k$$
$$\geq (1 + (\rho - 1)m) D_k \quad [\because m > 1, L_k \leq D_k]. \tag{19}$$

Then by (18) and (19), as well as the upper bounds for $\chi^{k,a}$ in Lemma 6 and for $F^{k,a}$ in (4), we have

$$(1+(\rho-1)m) D_k \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a} + \sum_i u_i n_i^{k,a} P_i$$
$$\Rightarrow (1+(\rho-1)m) D_k \leq \beta(1+\epsilon)\sum_i u_i \Big(\alpha_i^{k,a} + n_i^{k,a} P_i\Big)$$
$$[\because \epsilon > 0, \beta > 1]$$
$$\Rightarrow (1+(\rho-1)m) D_k \leq \beta(1+\epsilon)\sum_i u_i D_k \quad [\text{from (5)}]$$
$$\Rightarrow (1+(\rho-1)m) D_k \leq \beta(1+\epsilon) m D_k \quad \left[\because \sum_i u_i \leq m\right]$$
$$\Leftrightarrow 1 + (\rho-1)m \leq \beta(1+\epsilon) m$$

$$\Leftrightarrow \rho \leq \beta(1+\epsilon) + 1 - \frac{1}{m}$$
$$\Rightarrow \rho < \beta(1+\epsilon) + \left(\epsilon + \frac{1}{\epsilon}\right)\left(1 - \frac{1}{m}\right)$$
$$\left[\because m > 1, \epsilon + \frac{1}{\epsilon} \geq 2\right].$$

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 ϵ. 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 ϵ 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 $\beta + 2\sqrt{(\beta + 1 - (1/m))(1 - (1/m))}$ with $\epsilon = \sqrt{([1 - (1/m)]/[\beta + 1 - (1/m)])}$.

Proof: We rewrite the $\beta(1+\epsilon)+(\epsilon+(1/\epsilon))(1-(1/m))$ 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 $\beta+A+B$

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

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

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

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

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 $\rho \geq \beta + 2\sqrt{(\beta + 1 - (1/m))(1 - (1/m))}$, i.e., GEDF has a capacity augmentation bound of $\beta + 2\sqrt{(\beta + 1 - (1/m))(1 - (1/m))}$, 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 τ with constrained deadlines satisfies the following two conditions:

$$U_{\sum} \leq \frac{m}{\beta + 2\sqrt{\left(\beta + 1 - \frac{1}{m}\right)\left(1 - \frac{1}{m}\right)}}$$

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

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

C. Lower Capacity Augmentation Bound

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

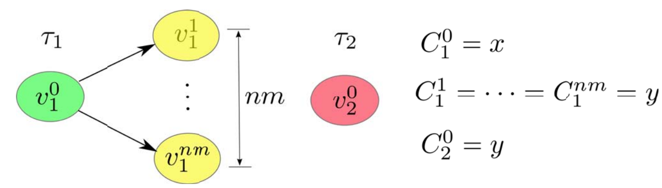


Fig. 7. Structure of the task set that demonstrates GEDF does not provide a capacity augmentation bound less than $[(\beta + \sqrt{\beta^2 + 4\beta})/2] + 1$.

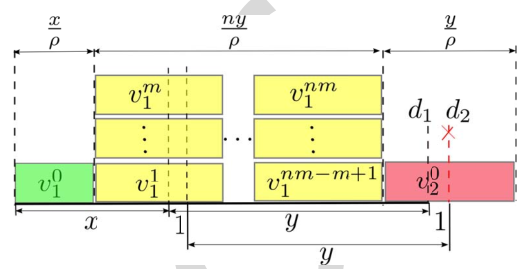


Fig. 8. Execution of the task set under GEDF at speed ρ.

The example is constructed as shown in Fig. 7. The task set contains two tasks. One task τ_1 is structured as a single vertex with workload x followed by nm vertices with workload y. Its critical path length L_1 is $x+y$ and so is its deadline. The period of τ_1 is set to be $\beta(x+y)$, and moreover, the utilization u_1 is set to be $m - 1$

$$m - 1 = \frac{x + nmy}{\beta(x + y)}. \tag{20}$$

The other task τ_2 has a single vertex with workload, deadline, and period equal to $x + y - (x/\rho)$, and thus the critical path length L_2 of τ_2 is $x + y - (x/\rho)$ and the utilization u_2 of τ_2 is 1.

Obviously, the necessity conditions (2) and (3) hold: $U_{\sum} = u_1 + u_2 \leq m$, $L_1 \leq D_1$ and $L_2 \leq D_2$. During the execution, τ_1 is released at the absolute time 0, and τ_2 is released at time $(x/\rho) + 1$. The execution is shown in Fig. 8.

We want to generate an example, so we want τ_2 to miss its deadline. In order for this to occur, we must have

$$x + y - \frac{x}{\rho} + 1 < \frac{ny + x + y - \frac{x}{\rho}}{\rho}. \tag{21}$$

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

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

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

$$\rho < \frac{(\beta + 2)nm + \sqrt{\left(\beta^2 + 4\beta\right)n^2m^2}}{2nm}$$
$$\Leftrightarrow \rho < \frac{\beta + \sqrt{\beta^2 + 4\beta}}{2} + 1. \tag{23}$$

So there exists an example for any speed-up ρ that satisfies the above conditions. Therefore, the capacity augmentation

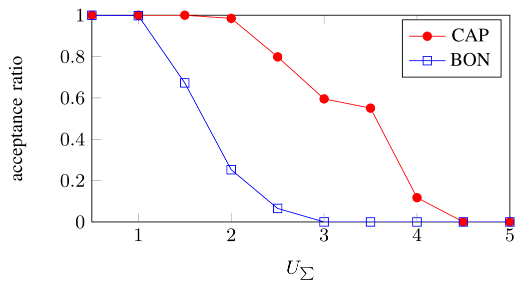


Fig. 9. $n = 20, m = 16, \beta = 2, p = 0.25$.

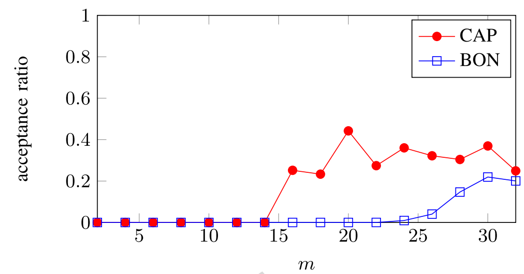


Fig. 10. $n = 20, U_\sum = 4, \beta = 2, p = 0.25$.

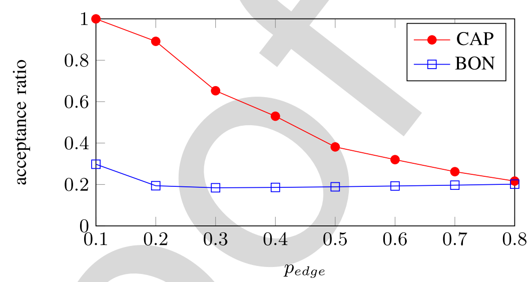


Fig. 11. $n = 20, m = 16, U_\sum = 2, \beta = 2.5$.

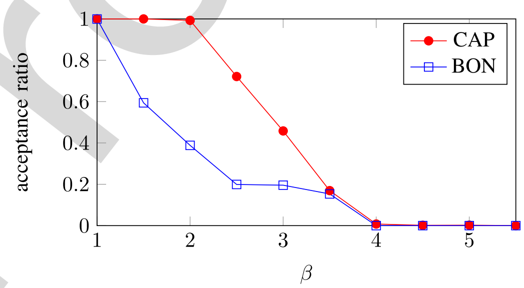


Fig. 12. $n = 20, m = 16, U_\sum = 2, p = 0.25$.

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:

$$\frac{2\beta + 4\sqrt{\beta + 1}}{\beta + \sqrt{\beta^2 + 4\beta} + 2}. \tag{24}$$

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 τ_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 $U_\sum$, 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 $U_\sum = 4$, 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 $U_\sum = 2$, $n = 20$, $m = 16$, and $\beta = 2.5$. 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 β (which determines the relative deadlines of tasks), with $U_\sum = 2$, $n = 20$, $m = 16$, and $p = 0.25$. For both tests, the schedulability ratio decreases when β increases. However, CAP can tolerate the increase of β 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

[1] (2018). *CilkPlus*. [Online]. Available: https://software.intel.com/en-us/intel-cilk-plus-support

[2] OpenMP Architecture Review Board. (2013). *OpenMP Application Program Interface, Version 4.0*. [Online]. Available: http://www.openmp.org/

[3] J. Sun, N. Guan, Y. Wang, Q. He, and W. Yi, "Real-time scheduling and analysis of OpenMP task systems with tied tasks," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Paris, France, 2017, pp. 92–103.

[4] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*, Sebastopol, CA, USA, O'Reilly Media, 2007.

[5] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *Proc. 31st IEEE Real Time Syst. Symp.*, San Diego, CA, USA, 2010, pp. 259–268.

[6] P. Axer *et al.*, "Response-time analysis of parallel fork-join workloads with real-time constraints," in *Proc. IEEE 25th Euromicro Conf. Real Time Syst. (ECRTS)*, 2013, pp. 215–224.

[7] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real Time Syst.*, vol. 49, no. 4, pp. 404–435, 2013.

[8] B. Andersson and D. de Niz, "Analyzing global-EDF for multiprocessor scheduling of parallel tasks," in *Proc. Int. Conf. Principles Distrib. Syst.*, 2012, pp. 16–30.

[9] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Techniques optimizing the number of processors to schedule multi-threaded tasks," in *Proc. IEEE 24th Euromicro Conf. Real Time Syst. (ECRTS)*, Pisa, Italy, 2012, pp. 321–330.

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

[11] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-time analysis of synchronous parallel tasks in multiprocessor systems," in *Proc. ACM 22nd Int. Conf. Real Time Netw. Syst.*, Versailles, France, 2014, p. 3.

[12] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *Proc. IEEE 33rd Real Time Syst. Symp. (RTSS)*, San Juan, PR, USA, 2012, pp. 63–72.

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

[14] J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of global EDF for parallel tasks," in *Proc. IEEE 25th Euromicro Conf. Real Time Syst. (ECRTS)*, 2013, pp. 3–13.

[15] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global EDF scheduling of directed acyclic graphs on multiprocessor systems," in *Proc. ACM 21st Int. Conf. Real Time Netw. Syst.*, Sophia Antipolis, France, 2013, pp. 287–296.

[16] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *Proc. IEEE 26th Euromicro Conf. Real Time Syst. (ECRTS)*, 2014, pp. 97–105.

[17] A. Saifullah *et al.*, "Parallel real-time scheduling of DAGs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3242–3252, Dec. 2014.

[18] J. Li *et al.*, "Analysis of federated and global scheduling for parallel real-time tasks," in *Proc. IEEE 26th Euromicro Conf. Real Time Syst. (ECRTS)*, Madrid, Spain, 2014, pp. 85–96.

[19] J. Li *et al.*, "Global EDF scheduling for parallel real-time tasks," *Real Time Syst.*, vol. 51, no. 4, pp. 395–439, 2015.

[20] X. Jiang, X. Long, N. Guan, and H. Wan, "On the decomposition-based global EDF scheduling of parallel real-time tasks," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Porto, Portugal, 2016, pp. 237–246.

[21] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo, "Schedulability analysis of conditional parallel task graphs in multicore systems," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 339–353, Feb. 2017.

[22] M. Qamhieh, L. George, and S. Midonnet, "A stretching algorithm for parallel real-time DAG tasks on multiprocessor systems," in *Proc. ACM 22nd Int. Conf. Real Time Netw. Syst.*, 2014, p. 13.

[23] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *Proc. IEEE 27th Euromicro Conf. Real Time Syst. (ECRTS)*, Lund, Sweden, 2015, pp. 211–221.

[24] J.-J. Chen, "Federated scheduling admits no constant speedup factors for constrained-deadline DAG task systems," *Real Time Syst.*, vol. 52, no. 6, pp. 833–838, 2016.

[25] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient multi-core scheduling for real-time DAG tasks," in *Proc. LIPIcs-Leibniz Int. Informat.*, vol. 76, 2017, p. 22.

[26] S. Baruah, "The federated scheduling of constrained-deadline sporadic DAG task systems," in *Proc. Design Autom. Test Europe Conf. Exhibit.*, Grenoble, France, 2015, pp. 1323–1328.

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

[28] S. Baruah, "The federated scheduling of systems of conditional sporadic DAG tasks," in *Proc. 12th Int. Conf. Embedded Softw.*, Amsterdam, The Netherlands, 2015, pp. 1–10.

[29] J. Li *et al.*, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real Time Syst.*, vol. 53, no. 5, pp. 760–811, 2017.

[30] J. Lelli, G. Lipari, D. Faggioli, and T. Cucinotta, "An efficient and scalable implementation of global EDF in Linux," in *Proc. 7th Int. Workshop Oper. Syst. Platforms Embedded Real Time Appl. (OSPERT)*, 2011, pp. 6–15.

[31] B. B. Brandenburg and J. H. Anderson, "On the implementation of global real-time schedulers," in *Proc. 30th IEEE Real Time Syst. Symp. (RTSS)*, Washington, DC, USA, 2009, pp. 214–224.

[32] A. Parri, A. Biondi, and M. Marinoni, "Response time analysis for G-EDF and G-DM scheduling of sporadic DAG-tasks with arbitrary deadline," in *Proc. ACM 23rd Int. Conf. Real Time Netw. Syst.*, 2015, pp. 205–214.

[33] D. Cordeiro *et al.*, "Random graph generation for scheduling simulations," in *Proc. 3rd Int. ICST Conf. Simulat. Tools Techn. (ICST)*, 2010, p. 60.

Jinghao Sun received the M.S. and Ph.D. degrees in computer science from the Dalian University of Technology, Dalian, China, in 2012.

He is an Associated Professor with Northeastern University, Shenyang, China. He was a Post-Doctoral Fellow with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, from 2016 to 2017, researching on scheduling algorithms for multicore real time systems. His current research interests include algorithms, schedulability analysis, and optimization methods.

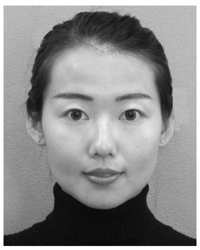
Nan Guan received the Ph.D. degree from Uppsala University, Uppsala, Sweden, in 2013.

He is currently an Assistant Professor with Hong Kong Polytechnic University, Hong Kong. His current research interests include safe-critical cyber-physical systems, real-time scheduling theory, and worst-case execution time analysis and formal verification techniques.

Dr. Guan was a recipient of the European Design Automation Association Outstanding Dissertation Award in 2014, the Best Paper Award of IEEE Real-Time Systems Symposium in 2009, the Best Paper Award of Design Automation and Test in Europe Conference in 2013, the Best Poster Award in the Ph.D. forum of IEEE International Parallel and Distributed Processing Symposium in 2012, and the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications in 2017.

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.

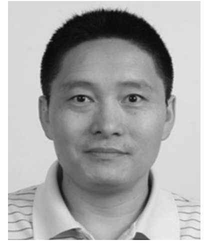


Shuangshuang Chang received the M.S. degree in computer technology from Northeastern University, Shenyang, China, in 2016, where she is currently pursuing the Ph.D. degree.

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 on the steering committee of ESWEEK, the annual joint event for major conferences in embedded systems areas. 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).

A Capacity Augmentation Bound for Real-Time Constrained-Deadline Parallel Tasks Under GEDF

Jinghao Sun, Nan Guan, Xu Jiang, Shuangshuang Chang, Zhishan Guo, Qingxu Deng, and Wang Yi, *Fellow, IEEE*

***Abstract*—Capacity augmentation bound is a widely used quantitative metric in theoretical studies of schedulability analysis for directed acyclic graph (DAG) parallel real-time tasks, which not only quantifies the suboptimality of the scheduling algorithms, but also serves as a simple linear-time schedulability test. Earlier studies on capacity augmentation bounds of the sporadic DAG task model were either restricted to a single DAG task or a set of tasks with implicit deadlines. In this paper, we consider parallel tasks with constrained deadlines under global earliest deadline first policy. We first show that it is impossible to obtain a constant bound for our problem setting, and derive both lower and upper bounds of the capacity augmentation bound as a function with respect to the maximum ratio of task period to deadline. Our upper bound is at most 1.47 times larger than the optimal one. We conduct experiments to compare the acceptance ratio of our capacity augmentation bound with the existing schedulability test also having linear-time complexity. The results show that our capacity augmentation bound significantly outperforms the existing linear-time schedulability test under different parameter settings.**

***Index Terms*—Capacity augmentation bound, directed acyclic graph (DAG), global earliest deadline first (GEDF), parallel tasks, real-time scheduling, schedulability analysis.**

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. This work was supported in part by the Research Grants Council of Hong Kong under Grant ECS 25204216 and Grant GRF 15204917, in part by the University Grants Committee of Hong Kong under Project 1-ZVJ2 through The Hong Kong Polytechnic University, in part by the National Nature Science Foundation under Grant CNS 1755965, in part by the National Nature Science Foundation of China under Grant 61672140 and Grant 61472072, and in part by the Fundamental Research Funds for the Central Universities under Grant N172304025. This paper was recommended by Conference Chairperson B. Brandenburg. *(Corresponding author: Nan Guan.)*

J. Sun is with the School of Computer Science and Engineering, Northeastern University, Shenyang 110004, China, and also with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: jhsun@mail.dlut.edu.cn).

N. Guan and X. Jiang are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: nan.guan@polyu.edu.hk; jiangxu617@163.com).

S. Chang and Q. Deng are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110004, China (e-mail: changs1393587345@sina.co; dengqx@mail.neu.edu.cn).

Z. Guo is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32611 USA, and also with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO 65409 USA (e-mail: guozh@mst.edu).

W. Yi is with the School of Computer Science and Engineering, Northeastern University, Shenyang 110004, China, and also with the Department of Information Technology, Uppsala University, 752 36 Uppsala, Sweden (e-mail: yi@it.uu.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2857362

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 parallelisim, but also intratask parallelisim 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.

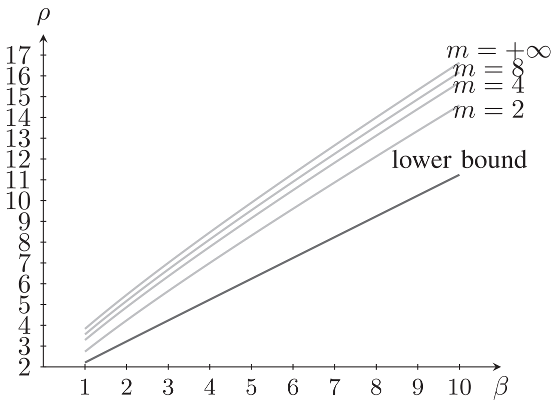


Fig. 1. Capacity bound as a function of β, and the red line represents the lower bound of capacity augmentation.

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 ρ if it can successfully schedule any task set τ on m cores of speed ρ as long as the compared scheduler can schedule τ 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 ρ if it can schedule any task set τ satisfying the following two conditions: a) the total utilization of τ is at most m/ρ 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 ρ, it is also guaranteed to have a resource augmentation bound of ρ. 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

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

where m is the number of processing cores and β 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 $(\beta + \sqrt{\beta^2 + 4\beta})/2 + 1$. Fig. 1 shows the figure of this capacity augmentation bound as a function of β.

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[1] 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 subcatagories: 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

[1]The test in [13] is for arbitrary-deadline DAG tasks, and thus also applicable to constrained-deadline DAG tasks considered in this paper.

TABLE I
STATE-OF-THE-ART RESOURCE AUGMENTATION BOUNDS (WITH RESPECT TO OPTIMAL SCHEDULING ALGORITHMS) AND CAPACITY AUGMENTATION BOUND FOR DAG TASKS (WHEN m IS INFINITELY LARGE)

<table>
<tr><td rowspan="2">DAG tasks</td><td colspan="2">global scheduling</td><td colspan="2">federated scheduling</td><td colspan="2">decomposition-based</td></tr>
<tr><td>resource</td><td>capacity</td><td>resource</td><td>capacity</td><td>resource</td><td>capacity</td></tr>
<tr><td>implicit deadline</td><td rowspan="3">2 for GEDF [13], [14], 3 for GDM [13]</td><td>$\frac{3+\sqrt{5}}{2}$ for GEDF, $2+\sqrt{3}$ for GRM [18]</td><td colspan="2">2 [18]</td><td colspan="2">[2,4) [20]</td></tr>
<tr><td>constrained deadline</td><td>$\beta + 2\sqrt{\beta+1}$ for GEDF (this work)</td><td colspan="4" rowspan="2">$\emptyset$</td></tr>
<tr><td>arbitrary deadline</td><td>$\emptyset$</td></tr>
</table>

number of cores. With respect to any optimal federated scheduling algorithm,[2] 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 $(11m/[3m - 3])$ for 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

For synchronous tasks with constrainted deadline, Chwa *et al.* [10] proposed an RTA-based analysis for GEDF scheduling algorithm, and Maia *et al.* [11] gave the anaylsis for GFP scheduling algorithm. Axer *et al.* [6] proposed an RTA-based analysis for fork-join tasks with arbitary deadline. Qamhieh *et al.* [15] gave an RTA-based analysis for GEDF scheduling of DAG-tasks with constrained deadline and a study of its sustainability. Parri *et al.* [32] proposed an RTA-based test for GEDF and GDM scheduling of DAG-tasks with arbitrary deadline. Melani *et al.* [21] proposed an RTA-based test for GEDF scheduling of conditional DAG-tasks with constrained deadline.

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 τ that consists of n tasks $\tau = \{\tau_1, \ldots, \tau_n\}$. Each task τ_k is associated with a *period* P_k and a *relative deadline* D_k, and its execution has a DAG structure. The xth subtask of task τ_k is represented by *vertex* v_k^x in the DAG. If there is a directed edge from vertex v_k^x to vertex v_k^y, then v_k^x is v_k^y's *predecessor*. A subtask cannot start its execution until the completion of all its predecessors. Each vertex v_k^x has its own *worst-case execution time* C_k^x.

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.

[2]An optimal federated scheduling may not be a good scheduling strategy compared with an optimal scheduling algorithm.

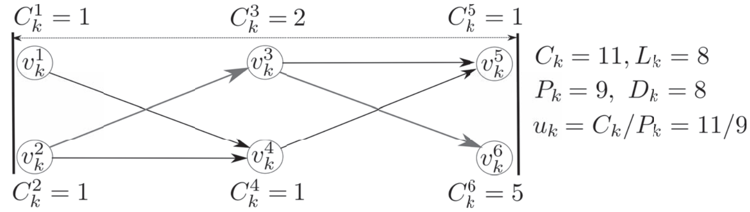


Fig. 2. Example DAG task τ_k with volume $C_k = 11$ and critical path length $L_k = 8$.

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 τ_k is the *volume* of τ_k

$$C_k = \sum_x C_k^x.$$

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

2) *Utilization:* We define the *utilization* u_k of a task τ_k as

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

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

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 τ_k as the longest path in τ_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 τ_k is $C_k = 11$, and the utilization of τ_k is $u_k = 11/9$. The critical path (marking in red) starts from vertex v_k^2, goes through v_k^3 and ends at vertex v_k^6, so the critical path length of the DAG task τ_k is $L_k = 1 + 2 + 5 = 8$.

A task τ_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 τ_k are the same as those of task τ_k.

Without loss of generality, $J_{k,a}$ denotes the ath job instance of task τ_k, and the xth vertex of $J_{k,a}$ is represented as $v_{k,a}^x$. 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

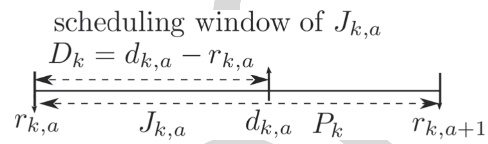


Fig. 3. Scheduling window $[r_{k,a}, d_{k,a}]$ of job $J_{k,a}$.

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_k^1 and v_k^2 are completely executed, and v_k^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 τ meet their deadlines.

Lemma 1 (Necessary Conditions for Schedulability [14]): A task set τ is not schedulable (by any scheduler) unless the following conditions hold.

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

$$\forall k : L_k \leq D_k. \quad (2)$$

2) The total utilization $U_\sum$ is smaller than the number of cores, i.e.,

$$U_\sum \leq m. \quad (3)$$

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

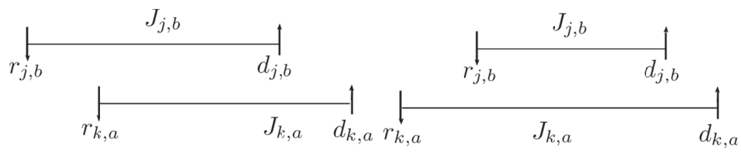


Fig. 4. Two types of jobs that may interfere with $J_{k,a}$. (a) $J_{j,b}$ is a carry-in job of $J_{k,a}$. (b) $J_{j,b}$ is a fall-in job of $J_{k,a}$.

exclusively on sufficiently many cores. If (3) is violated, then in the long term the worst-case workload of the system exceeds the processing capacity provided by the platform, and thus the backlog will increase infinitely which leads to deadline misses.

A scheduling algorithm S has a *capacity augmentation bound* ρ if any task set τ satisfying the following conditions is schedulable by S: 1) $\forall k : L_k \le D_k/\rho$ and 2) $U_{\sum} \le m/\rho$. The concept of *capacity augmentation bound* can be equivalently stated as follows [14] and [18]:

Definition 1 (Capacity Augmentation Bound for DAG Task System): A scheduling algorithm S has a *capacity augmentation bound* ρ if it can always schedule DAG task set τ on m cores of speed ρ as long as τ satisfies the above necessary conditions (2) and (3).

A scheduling algorithm with a smaller ρ is preferable and when $\rho = 1$ the scheduling algorithm S is optimal.

B. Overall Analysis Outline

The overall intuition behind the capacity bound analysis is to derive a sufficient condition, under which every released job can be successfully scheduled by GEDF on cores with speed ρ. More precisely, for each job $J_{k,a}$ under analysis, we derive a lower bound of the multicore resource that must be utilized to execute tasks in the scheduling window $[r_{k,a}, d_{k,a}]$ of $J_{k,a}$, and meanwhile, we derive an upper bound of the workload that must be executed by GEDF during the scheduling window $[r_{k,a}, d_{k,a}]$ of $J_{k,a}$. A sufficient condition for successfully scheduling tasks is that the resource's lower bound is larger than the workload's upper bound for all jobs. As we know that the lower resource bound increases with the core speed ρ and the upper workload bound decreases with the core speed ρ, we aim to find the minimum speed ρ to make the sufficient condition hold. Such a minimum speed ρ is the capacity augmentation bound as shown in Definition 1.

In the following, the upper workload bound is analyzed in Sections IV-A and V-A. Moreover, the lower resource bound is given in Section IV-B. Determining the infimum of speed ρ is given in Section V-B.

IV. Preliminary Results

In this section, we introduce some concepts and properties that will be useful in deriving the capacity augmentation bound in the next section.

A. Interference

Suppose we are analyzing the schedulability of an arbitrary job $J_{k,a}$, the ath instance of task τ_k, under GEDF scheduling. When analyzing $J_{k,a}$, we assume that all the other jobs can meet their deadlines. Another job $J_{j,b}$ of τ_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., $d_{j,b} \le d_{k,a}$.

For any task τ_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.,$r_{j,b} < r_{k,a} \wedge d_{j,b} \le d_{k,a}$ [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., $r_{j,b} \ge r_{k,a} \wedge d_{j,b} \le d_{k,a}$ [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 $n_j^{k,a}$ be the number of $J_{k,a}$'s fall-in jobs that are released from the task τ_j (see an example in Fig. 5). The total amount

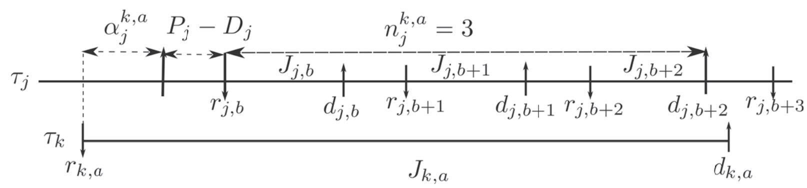


Fig. 5. Number of $J_{k,a}$'s fall-in jobs from τ_j is $n_j^{k,a} = 3$.

of the fall-in work of $J_{k,a}$ is upper bounded by

$$F^{k,a} \le \sum_i n_i^{k,a} C_i = \sum_i u_i n_i^{k,a} P_i. \tag{4}$$

Definition 4 (Remaining Window Length): Let $J_{j,b}$ be a carry-in job from task τ_j for the analyzed job $J_{k,a}$, the remaining window length of τ_j is defined as

$$\alpha_j^{k,a} = d_{j,b} - r_{k,a}.$$

Obviously, $\alpha_j^{k,a} \le D_j$ [see Fig. 4(a)]. Moreover, as shown in Fig. 5, the following inequality holds:

$$\begin{aligned} D_k &\ge \alpha_j^{k,a} + P_j - D_j + \left(n_j^{k,a} - 1\right) P_j + D_j \\ &= \alpha_j^{k,a} + n_j^{k,a} P_j. \end{aligned} \tag{5}$$

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 ρ. The following lemmas are given in [14].

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

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

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

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

Proof: We first prove that the accumulated length of incomplete intervals in $\mathcal{I}$, denoted by x, is no more than L_k/ρ. 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 $\mathcal{I}$. 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 $\mathcal{I}$ is at most L_k/ρ. By Lemma 3, the total amount of work done during $\mathcal{I}$ is at least $\rho m|\mathcal{I}| - (m-1)L_k$. ■

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 ϵ, 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 ϵ 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 ρ-speed cores. First, for the cores with speed $\rho \ge 1$, a straightforward bound for carry-in work of the analyzed job $J_{k,a}$ is as follows.

Lemma 5: If the core speed $\rho \ge 1$, the carry-in work $\chi^{k,a}$ for job $J_{k,a}$ is bounded by

$$\chi^{k,a} \le \beta \sum_i u_i D_i. \tag{6}$$

Proof: Using $\mathcal{J}_1$ to denote the set of carry-in jobs of $J_{k,a}$ that are unfinished at time $r_{k,a}$, then we have

$$\begin{aligned} \chi^{k,a} &\le \sum_{J_{j,b} \in \mathcal{J}_1} u_j P_j \\ &\le \beta \sum_{J_{j,b} \in \mathcal{J}_1} u_j D_j \quad \left[\because \beta = \max_i \left\{\frac{P_i}{D_i}\right\}\right] \\ &\le \beta \sum_i u_i D_i. \end{aligned}$$

The last step of the above inequality is because that each constrained-deadline task τ_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 ρ strictly larger than 1, by representing the infimum of core speed ρ 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

$$\chi^{k,a} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a} \tag{7}$$

where

$$\rho(\epsilon) = \beta(1+\epsilon) + \left(\epsilon + \frac{1}{\epsilon}\right)\left(1 - \frac{1}{m}\right). \tag{8}$$

(Recall that $\alpha_i^{k,a}$ is the remaining window length of task τ_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 very 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 $J_{j,b}$ released earlier than $J_{k,a}$ satisfies

$$\chi^{j,b} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{j,b}. \tag{9}$$

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

$$\begin{aligned}\chi^{k,a} &\leq \beta \sum_j u_j D_j \\ &< \beta(1+\epsilon)\sum_j u_j \alpha_j^{k,a} \quad \left[\because \alpha_j^{k,a} > \frac{D_j}{1+\epsilon}\right].\end{aligned}$$

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

$$\Delta \geq \frac{\epsilon}{1+\epsilon} D_j. \tag{10}$$

On the other hand, we have (see Fig. 6 for intuition)

$$\begin{aligned}\Delta &\geq \alpha_i^{j,b} + P_i - D_i + n_i^{\Delta} P_i + D_i - \alpha_i^{k,a} \\ &\geq \alpha_i^{j,b} + n_i^{\Delta} P_i + P_i - \alpha_i^{k,a}\end{aligned} \tag{11}$$

where n_i^{Δ} 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^{Δ}, is at least

$$W^{\Delta} \geq \rho m \Delta - (m-1)L_j. \tag{12}$$

The work of W^{Δ} comes from three sets of jobs.

1) $\mathcal{J}_A$: the set of carry-in jobs of $J_{j,b}$.
2) $\mathcal{J}_B$: the set of carry-in jobs of $J_{k,a}$.
3) $\mathcal{J}_C$: the set of jobs that entirely fall in $[r_{j,b}, r_{k,a}]$.

For example, in Fig. 6, $\mathcal{J}_A = \{J_{i,c}, J_{l,d}\}$ (in red rectangles), $\mathcal{J}_B = \{J_{i,c+2}, J_{l,d}\}$ (in blue rectangles) and $\mathcal{J}_C = \{J_{i,c+1}\}$ (in green rectangles). Obviously, $(\mathcal{J}_A \cup \mathcal{J}_B) \cap \mathcal{J}_C = \emptyset$, and in general $\mathcal{J}_A \cap \mathcal{J}_B \neq \emptyset$.

Let $\mathcal{J}_A' = \mathcal{J}_A - \mathcal{J}_B$. We use W_x to denote the total amount of work done by jobs in $\mathcal{J}_x$ (for $x = A', A, B, C$), the total amount of work W^{Δ} done during $[r_{j,b}, r_{k,a}]$ can be divided into three parts

$$W^{\Delta} = W_{A'} + W_B + W_C. \tag{13}$$

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 $\chi^{j,b}$ of $J_{j,b}$, i.e., $W_{A'} \leq \chi^{j,b}$, and by the inductive hypothesis (9), we have

$$W_{A'} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{j,b}. \tag{14}$$

Upper Bound of W_B: We observe that the total amount of work by the carry-in jobs of $J_{k,a}$, denoted by $C^{k,a}$ 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

$$C^{k,a} \geq W_B + \chi^{k,a}. \tag{15}$$

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

$$W_B \leq \sum_i u_i P_i - \chi^{k,a}. \tag{16}$$

Upper Bound of W_C: For each $\tau_i \in \tau$, recall that n_i^{Δ} 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 $\mathcal{J}_C$ can be calculated as

$$W_C = \sum_i u_i n_i^{\Delta} P_i. \tag{17}$$

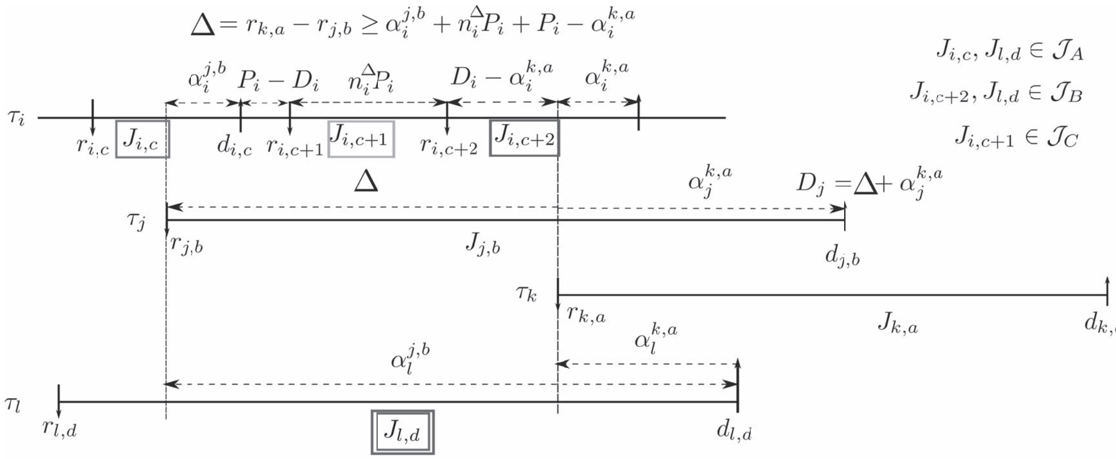


Fig. 6. Illustration for the proof of Lemma 6.

Putting (13), (14), (16), and (17) together, we have

$$
\begin{aligned}
W^{\Delta} &\leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{j,b} + \sum_i u_i n_i^{\Delta} P_i + \sum_i u_i P_i - \chi^{k,a} \\
&\leq \beta(1+\epsilon)\sum_i u_i\Big(\alpha_i^{j,b} + n_i^{\Delta} P_i + P_i\Big) - \chi^{k,a} \\
&\quad [\because \epsilon > 0, \beta > 1]
\end{aligned}
$$

and by (12), we have

$$
\begin{aligned}
\chi^{k,a} &\leq \beta(1+\epsilon)\sum_i u_i\Big(\alpha_i^{j,b} + n_i^{\Delta} P_i + P_i\Big) \\
&\quad - \rho m \Delta + (m-1) L_j \\
&\leq \beta(1+\epsilon)\sum_i u_i\Big(\Delta + \alpha_i^{k,a}\Big) - \rho m \Delta \\
&\quad + (m-1) L_j \quad [\because \ (11)]
\end{aligned}
$$

and since $\sum_i u_i \leq m$ and $L_j \leq D_j$, we have

$$
\chi^{k,a} \leq \beta(1+\epsilon)\left(m\Delta + \sum_i u_i \alpha_i^{k,a}\right) - \rho m \Delta + (m-1) D_j
$$

and by $\Delta \geq (\epsilon/[1+\epsilon]) D_j$, we have

$$
\begin{aligned}
\chi^{k,a} &\leq (\beta(1+\epsilon) - \rho) m \Delta + (m-1)\left(\epsilon + \frac{1}{\epsilon}\right)\Delta \\
&\quad + \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a}
\end{aligned}
$$

and since $\rho \geq \beta(1+\epsilon) + (\epsilon + (1/\epsilon))(1 - (1/m))$, we have

$$
\begin{aligned}
\chi^{k,a} &\leq \left(\epsilon + \frac{1}{\epsilon}\right)(1-m)\Delta + \left(\epsilon + \frac{1}{\epsilon}\right)(m-1)\Delta \\
&\quad + \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a}
\end{aligned}
$$

by which we finally get $\chi^{k,a} \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a}$. ■

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 $F^{k,a}$ by (4), and Lemma 6 bounds the carry-in work $\chi^{k,a}$, 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 $\beta = \max_i\{(P_i/D_i)\}$.

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

$$
W \leq \chi^{k,a} + F^{k,a}. \tag{18}
$$

Since $J_{k,a}$ has not finished at its absolute deadline $d_{k,a}$, by Lemma 4, we have

$$
\begin{aligned}
W &\geq \rho m D_k - (m-1) L_k \\
&\geq (1 + (\rho - 1) m) D_k \quad [\because m > 1, L_k \leq D_k]. \qquad (19)
\end{aligned}
$$

Then by (18) and (19), as well as the upper bounds for $\chi^{k,a}$ in Lemma 6 and for $F^{k,a}$ in (4), we have

$$
\begin{aligned}
&(1 + (\rho-1)m) D_k \leq \beta(1+\epsilon)\sum_i u_i \alpha_i^{k,a} + \sum_i u_i n_i^{k,a} P_i \\
&\Rightarrow (1 + (\rho-1)m) D_k \leq \beta(1+\epsilon)\sum_i u_i\Big(\alpha_i^{k,a} + n_i^{k,a} P_i\Big) \\
&\quad [\because \epsilon > 0, \beta > 1] \\
&\Rightarrow (1 + (\rho-1)m) D_k \leq \beta(1+\epsilon)\sum_i u_i D_k \quad [\text{from } (5)] \\
&\Rightarrow (1 + (\rho-1)m) D_k \leq \beta(1+\epsilon) m D_k \quad \left[\because \sum_i u_i \leq m\right] \\
&\Leftrightarrow 1 + (\rho-1)m \leq \beta(1+\epsilon) m
\end{aligned}
$$

$$\Leftrightarrow \rho \le \beta(1+\epsilon) + 1 - \frac{1}{m}$$

$$\Rightarrow \rho < \beta(1+\epsilon) + \left(\epsilon + \frac{1}{\epsilon}\right)\left(1 - \frac{1}{m}\right)$$

$$\left[\because m > 1, \epsilon + \frac{1}{\epsilon} \ge 2\right].$$

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

Note that the capacity augmentation bound in Lemma 7 contains an open variable ϵ. 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 ϵ 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 $\beta + 2\sqrt{(\beta + 1 - (1/m))(1 - (1/m))}$ with $\epsilon = \sqrt{([1 - (1/m)]/[\beta + 1 - (1/m)])}$.

Proof: We rewrite the $\beta(1+\epsilon)+(\epsilon+(1/\epsilon))(1-(1/m))$ 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 $A+B \ge 2\sqrt{AB}$, we know the lower bound of $\beta+A+B$

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

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

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

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

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 $\rho \ge \beta + 2\sqrt{(\beta + 1 - (1/m))(1 - (1/m))}$, i.e., GEDF has a capacity augmentation bound of $\beta + 2\sqrt{(\beta + 1 - (1/m))(1 - (1/m))}$, 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 τ with constrained deadlines satisfies the following two conditions:

$$U_{\sum} \le \frac{m}{\beta + 2\sqrt{\left(\beta + 1 - \frac{1}{m}\right)\left(1 - \frac{1}{m}\right)}}$$

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

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

C. Lower Capacity Augmentation Bound

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

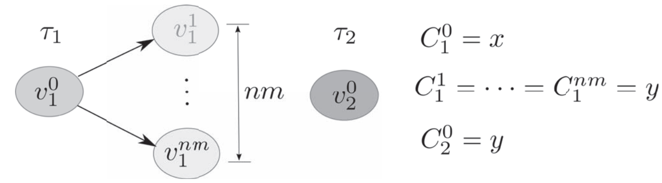


Fig. 7. Structure of the task set that demonstrates GEDF does not provide a capacity augmentation bound less than $[(\beta + \sqrt{\beta^2 + 4\beta})/2] + 1$.

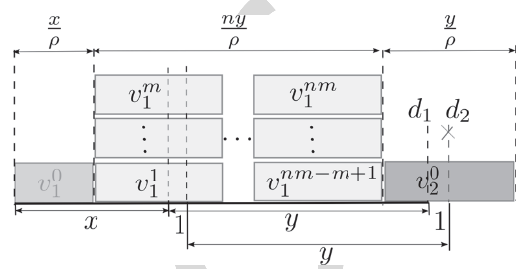


Fig. 8. Execution of the task set under GEDF at speed ρ.

The example is constructed as shown in Fig. 7. The task set contains two tasks. One task τ_1 is structured as a single vertex with workload x followed by nm vertices with workload y. Its critical path length L_1 is $x+y$ and so is its deadline. The period of τ_1 is set to be $\beta(x+y)$, and moreover, the utilization u_1 is set to be $m - 1$

$$m - 1 = \frac{x + nmy}{\beta(x+y)}. \tag{20}$$

The other task τ_2 has a single vertex with workload, deadline, and period equal to $x + y - (x/\rho)$, and thus the critical path length L_2 of τ_2 is $x + y - (x/\rho)$ and the utilization u_2 of τ_2 is 1.

Obviously, the necessity conditions (2) and (3) hold: $U_{\sum} = u_1 + u_2 \le m$, $L_1 \le D_1$ and $L_2 \le D_2$. During the execution, τ_1 is released at the absolute time 0, and τ_2 is released at time $(x/\rho) + 1$. The execution is shown in Fig. 8.

We want to generate an example, so we want τ_2 to miss its deadline. In order for this to occur, we must have

$$x + y - \frac{x}{\rho} + 1 < \frac{ny + x + y - \frac{x}{\rho}}{\rho}. \tag{21}$$

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

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

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

$$\rho < \frac{(\beta+2)nm + \sqrt{\left(\beta^2 + 4\beta\right)n^2m^2}}{2nm}$$

$$\Leftrightarrow \rho < \frac{\beta + \sqrt{\beta^2 + 4\beta}}{2} + 1. \tag{23}$$

So there exists an example for any speed-up ρ that satisfies the above conditions. Therefore, the capacity augmentation

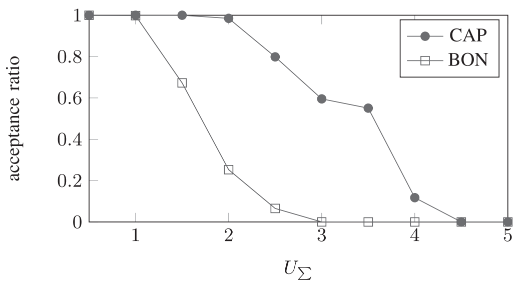


Fig. 9. $n = 20, m = 16, \beta = 2, p = 0.25$.

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:

$$\frac{2\beta + 4\sqrt{\beta + 1}}{\beta + \sqrt{\beta^2 + 4\beta} + 2}. \tag{24}$$

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 τ_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 $U_{\sum}$, 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 $U_{\sum} = 4$, 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.

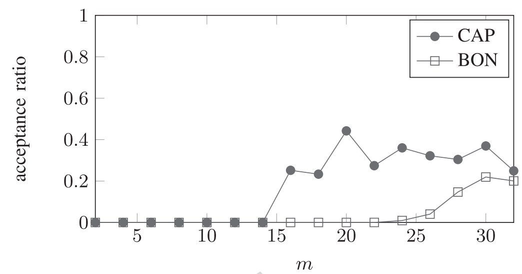


Fig. 10. $n = 20, U_{\sum} = 4, \beta = 2, p = 0.25$.

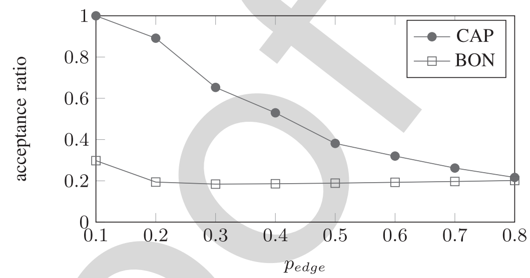


Fig. 11. $n = 20, m = 16, U_{\sum} = 2, \beta = 2.5$.

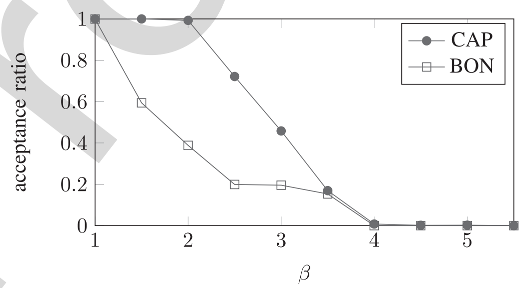


Fig. 12. $n = 20, m = 16, U_{\sum} = 2, p = 0.25$.

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 $U_{\sum} = 2$, $n = 20$, $m = 16$, and $\beta = 2.5$. 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 β (which determines the relative deadlines of tasks), with $U_{\sum} = 2$, $n = 20$, $m = 16$, and $p = 0.25$. For both tests, the schedulability ratio decreases when β increases. However, CAP can tolerate the increase of β 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

[1] (2018). *CilkPlus*. [Online]. Available: https://software.intel.com/en-us/intel-cilk-plus-support

[2] OpenMP Architecture Review Board. (2013). *OpenMP Application Program Interface, Version 4.0*. [Online]. Available: http://www.openmp.org/

[3] J. Sun, N. Guan, Y. Wang, Q. He, and W. Yi, "Real-time scheduling and analysis of OpenMP task systems with tied tasks," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Paris, France, 2017, pp. 92–103.

[4] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*, Sebastopol, CA, USA, O'Reilly Media, 2007.

[5] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *Proc. 31st IEEE Real Time Syst. Symp.*, San Diego, CA, USA, 2010, pp. 259–268.

[6] P. Axer *et al.*, "Response-time analysis of parallel fork-join workloads with real-time constraints," in *Proc. IEEE 25th Euromicro Conf. Real Time Syst. (ECRTS)*, 2013, pp. 215–224.

[7] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real Time Syst.*, vol. 49, no. 4, pp. 404–435, 2013.

[8] B. Andersson and D. de Niz, "Analyzing global-EDF for multiprocessor scheduling of parallel tasks," in *Proc. Int. Conf. Principles Distrib. Syst.*, 2012, pp. 16–30.

[9] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Techniques optimizing the number of processors to schedule multi-threaded tasks," in *Proc. IEEE 24th Euromicro Conf. Real Time Syst. (ECRTS)*, Pisa, Italy, 2012, pp. 321–330.

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

[11] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-time analysis of synchronous parallel tasks in multiprocessor systems," in *Proc. ACM 22nd Int. Conf. Real Time Netw. Syst.*, Versailles, France, 2014, p. 3.

[12] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *Proc. IEEE 33rd Real Time Syst. Symp. (RTSS)*, San Juan, PR, USA, 2012, pp. 63–72.

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

[14] J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of global EDF for parallel tasks," in *Proc. IEEE 25th Euromicro Conf. Real Time Syst. (ECRTS)*, 2013, pp. 3–13.

[15] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global EDF scheduling of directed acyclic graphs on multiprocessor systems," in *Proc. ACM 21st Int. Conf. Real Time Netw. Syst.*, Sophia Antipolis, France, 2013, pp. 287–296.

[16] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *Proc. IEEE 26th Euromicro Conf. Real Time Syst. (ECRTS)*, 2014, pp. 97–105.

[17] A. Saifullah *et al.*, "Parallel real-time scheduling of DAGs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3242–3252, Dec. 2014.

[18] J. Li *et al.*, "Analysis of federated and global scheduling for parallel real-time tasks," in *Proc. IEEE 26th Euromicro Conf. Real Time Syst. (ECRTS)*, Madrid, Spain, 2014, pp. 85–96.

[19] J. Li *et al.*, "Global EDF scheduling for parallel real-time tasks," *Real Time Syst.*, vol. 51, no. 4, pp. 395–439, 2015.

[20] X. Jiang, X. Long, N. Guan, and H. Wan, "On the decomposition-based global EDF scheduling of parallel real-time tasks," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Porto, Portugal, 2016, pp. 237–246.

[21] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo, "Schedulability analysis of conditional parallel task graphs in multicore systems," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 339–353, Feb. 2017.

[22] M. Qamhieh, L. George, and S. Midonnet, "A stretching algorithm for parallel real-time DAG tasks on multiprocessor systems," in *Proc. ACM 22nd Int. Conf. Real Time Netw. Syst.*, 2014, p. 13.

[23] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *Proc. IEEE 27th Euromicro Conf. Real Time Syst. (ECRTS)*, Lund, Sweden, 2015, pp. 211–221.

[24] J.-J. Chen, "Federated scheduling admits no constant speedup factors for constrained-deadline DAG task systems," *Real Time Syst.*, vol. 52, no. 6, pp. 833–838, 2016.

[25] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient multi-core scheduling for real-time DAG tasks," in *Proc. LIPIcs-Leibniz Int. Informat.*, vol. 76, 2017, p. 22.

[26] S. Baruah, "The federated scheduling of constrained-deadline sporadic DAG task systems," in *Proc. Design Autom. Test Europe Conf. Exhibit.*, Grenoble, France, 2015, pp. 1323–1328.

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

[28] S. Baruah, "The federated scheduling of systems of conditional sporadic DAG tasks," in *Proc. 12th Int. Conf. Embedded Softw.*, Amsterdam, The Netherlands, 2015, pp. 1–10.

[29] J. Li *et al.*, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real Time Syst.*, vol. 53, no. 5, pp. 760–811, 2017.

[30] J. Lelli, G. Lipari, D. Faggioli, and T. Cucinotta, "An efficient and scalable implementation of global EDF in Linux," in *Proc. 7th Int. Workshop Oper. Syst. Platforms Embedded Real Time Appl. (OSPERT)*, 2011, pp. 6–15.

[31] B. B. Brandenburg and J. H. Anderson, "On the implementation of global real-time schedulers," in *Proc. 30th IEEE Real Time Syst. Symp. (RTSS)*, Washington, DC, USA, 2009, pp. 214–224.

[32] A. Parri, A. Biondi, and M. Marinoni, "Response time analysis for G-EDF and G-DM scheduling of sporadic DAG-tasks with arbitrary deadline," in *Proc. ACM 23rd Int. Conf. Real Time Netw. Syst.*, 2015, pp. 205–214.

[33] D. Cordeiro *et al.*, "Random graph generation for scheduling simulations," in *Proc. 3rd Int. ICST Conf. Simulat. Tools Techn. (ICST)*, 2010, p. 60.

Jinghao Sun received the M.S. and Ph.D. degrees in computer science from the Dalian University of Technology, Dalian, China, in 2012.

He is an Associated Professor with Northeastern University, Shenyang, China. He was a Post-Doctoral Fellow with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, from 2016 to 2017, researching on scheduling algorithms for multicore real time systems. His current research interests include algorithms, schedulability analysis, and optimization methods.

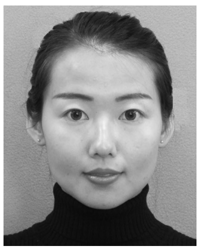
Nan Guan received the Ph.D. degree from Uppsala University, Uppsala, Sweden, in 2013.

He is currently an Assistant Professor with Hong Kong Polytechnic University, Hong Kong. His current research interests include safe-critical cyber-physical systems, real-time scheduling theory, and worst-case execution time analysis and formal verification techniques.

Dr. Guan was a recipient of the European Design Automation Association Outstanding Dissertation Award in 2014, the Best Paper Award of IEEE Real-Time Systems Symposium in 2009, the Best Paper Award of Design Automation and Test in Europe Conference in 2013, the Best Poster Award in the Ph.D. forum of IEEE International Parallel and Distributed Processing Symposium in 2012, and the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications in 2017.

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.

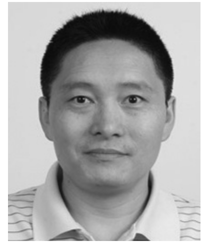


Shuangshuang Chang received the M.S. degree in computer technology from Northeastern University, Shenyang, China, in 2016, where she is currently pursuing the Ph.D. degree.

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 on the steering committee of ESWEEK, the annual joint event for major conferences in embedded systems areas. 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).