Analyzing GEDF Scheduling for Parallel Real-Time Tasks with Arbitrary Deadlines

Xu Jiang[1], Nan Guan[1], Di Liu[2], Weichen Liu[3]

[1]The Hong Kong Polytechnic University, Hong Kong [2]Yunnan University, China
[3]Nanyang Technological University, Singapore

***Abstract*—Real-time and embedded systems are shifting from single-core to multi-core processors, on which software must be parallelized to fully utilize the computation capacity of hardware. Recently much work has been done on real-time scheduling of *parallel* tasks modeled as directed acyclic graphs (DAG). However, most of these studies assume tasks to have implicit or constrained deadlines. Much less work considered the general case of arbitrary deadlines (i.e., the relative deadline is allowed to be larger than the period), which is more difficult to analyze due to intra-task interference among jobs. In this paper, we study the analysis of Global Earliest Deadline First (GEDF) scheduling for DAG parallel tasks with arbitrary deadlines. We develop new analysis techniques for GEDF scheduling of a single DAG task, which not only outperform the state-of-the-art in general evidenced by empirical evaluation, but also guarantee a better capacity augmentation bound 2.41 (the best known result is 2.5). The proposed analysis techniques are also extended to and evaluated with the case of multiple DAG tasks using the federated scheduling approach.**

I. Introduction

As the growing prevalence of multi-core processors, parallel programming models (such as OpenMP [1]) have drawn increasing attentions in real-time systems, to exploit the intra-task parallelism such that rapidly increasing performance requirements and more cruel real-time constraints can be satisfied. Existing scheduling and analysis techniques for sequential real-time tasks are hard to migrate to the parallel workload setting. New scheduling and analysis techniques are required to efficiently deploy parallel real-time tasks on multi-cores.

Directed Acyclic Graph (DAG) is a general model to represent parallel workload. Recently, many scheduling algorithms and analysis techniques for the DAG real-time task model [2]–[9] have been studied, most of which assume the tasks to have *implicit* deadlines (deadlines equal to periods) or *constrained* deadlines (deadlines no larger than periods). Much less work has been done for the general case of *arbitrary* deadlines (deadlines can be $(<, =, >)$ periods). With arbitrary deadlines, a new job may be released before previously released jobs are finished, and thus the analysis has to address the intra-task interference among jobs. Therefore, the analysis of real-time scheduling of DAG tasks with arbitrary deadlines is more difficult than that with implicit or constrained deadlines.

In this paper, we study real-time scheduling analysis of DAG tasks with *arbitrary* deadlines on multi-core processors. We develop new analysis techniques for scheduling a *single* DAG task under Global Earlies Deadline First (GEDF). Our new analysis techniques outperform the state-of-the-art with both theoretical metrics and experimental evaluations:

- We prove a *capacity augmentation bound* of $\sqrt{2}+1 \approx 2.41$ for GEDF. Capacity augmentation bound is a widely used quantitative metric for real-time scheduling of DAG tasks [2], [4], [6]–[8], which can not only quantify the sub-optimality of scheduling algorithms but also serves as a straightforward schedulability test. The best known capacity augmentation before our work is 2.5 [10].
- We conduct comprehensive experiments to evaluate the performance of our techniques, in comparison with the state-of-the-art approaches. The experimental results show that our techniques consistently outperform the state-of-the-art under different parameter settings.

Furthermore, we extend our analysis techniques to schedule *multiple* DAG tasks with arbitrary deadlines using the federated scheduling approach [4]. We also evaluate the real-time performance of our method in comparison with the state-of-the-art methods. The results indicate that our method also outperforms other approaches for multiple DAG tasks.

A. Related Work

Two theoretical metrics are used to evaluate performance of real-time scheduling algorithms and schedulability analysis for parallel tasks: the *speed up factor* [2]–[4] and the capacity augmentation bound. Capacity augmentation bound is a stronger metric than speed up factor in the sense that (1) the capacity augmentation bound also serves as a schedulability test [2] while the speed up factor does not [11], and (2) a scheduling algorithm having a capacity augmentation bound β implies a speed up factor β [2]. For scheduling a single DAG task, Graham [12] presented an upper bound of the response time for scheduling a DAG task with constrained deadline with a tight speed up factor of $2 - 1/m$. Baruah et.al [10] proved a tight speed up factor of $2 - 1/m$ and a capacity augmentation bound of 2.5 for scheduling a DAG task with arbitrary deadline under GEDF. Two schedulability tests are also provided in [10], with polynomial and pseudo-polynomial time respectively. For scheduling multiple DAG tasks, much work has been done with the implicit-deadline and constrained-deadline setting [4], [6], [7], [13]–[16]. Two tight capacity augmentation bounds of $\frac{3+\sqrt{5}}{2}$ and 2 are proved for scheduling DAG tasks with implicit deadlines under GEDF and federated scheduling respectively [4]. Baruah proved a speed up factor of $3 - 1/m$ with respect to any optimal federated scheduling algorithm for DAG tasks with constrained deadlines [8]. For DAG tasks with arbitrary deadlines, Baruah presented a schedulability test under federated scheduling in [7] and proved a speed up factor of $4 - 2/m$ with respect to any optimal federated scheduling algorithm. In both [2] and [3], a tight speed up factor of $2-1/m$ is proved for GEDF, and in [3] they also provided a schedulability test with polynomial time for GEDF.

II. Preliminary

For simplicity of presentation, we first consider the analysis of a single task τ scheduled on m identical processors under GEDF. Later in Section III-E we will extend our consideration to the case of multiple tasks.

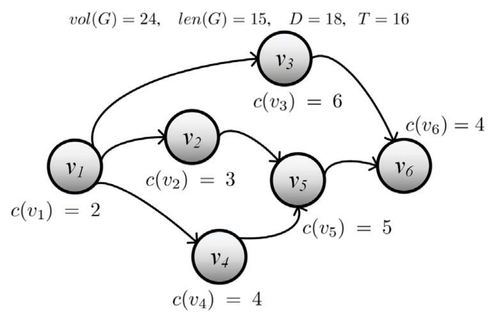


Fig. 1. An example of a DAG task τ.

A. Task Model

A task $\tau = \langle G, D, T \rangle$ has a *relative deadline* D, a *period* T and a workload structure modeled by a Directed Acyclic Graph (DAG) $G = \langle V, E \rangle$, where V is the set of vertices and E is the set of edges in G. A vertex $u \in V$ is characterized by a Worst-case Execution Time (WCET) $c(u)$. We use $vol(G)$ to denote the total WCET of all vertices in τ's DAG G: $vol(G) = \sum_{u \in V} c(u)$. The *utilization* U of τ is defined as $U = \frac{vol(G)}{T}$.

Each edge $(u, v) \in E$ represents the precedence relation between vertices u and v, where u is a *predecessor* of v, and v is a *successor* of u. We assume each DAG has a unique head vertex (with no predecessor) and a unique tail vertex (with no successor). This assumption does not limit the expressiveness of our model since one can always add a dummy head/tail vertex to a DAG with multiple entry/exit points. A *path* in a DAG task is a sequence of vertices $\pi = \{v_1, v_2, ...v_k\}$ from the head vertex v_1 to the tail vertex v_k, where v_i is a predecessor of v_{i+1}. The length of each path π is denoted by $len(\pi)$. We use $len(G)$ to denote the longest length of all paths in G.

Task τ generates a potentially infinite sequence of jobs, which inherit the DAG workload structure of the task. We use J_k to denote the k^{th} job of τ. The *release time* of a job J_k is denoted by r_k and its *finish time* is denoted by f_k. A job J_k is said to be *pending* at time t if it has been released but not finished by t. At runtime, we say a vertex (of a job J_k) is *eligible* at some time point if all its predecessors (of the same job J_k) have been finished and thus it can immediately execute if there are available processors. The release times of two successive jobs are separated by at least T time units and each job J_k must be finished before its absolute deadline d_k which is calculated by $r_k + D$. Fig.1 shows an example of DAG task τ with 6 vertices. We can compute $vol(G) = 24$ and $len(G) = 15$ (the longest path is $\{v_1, v_4, v_5, v_6\}$).

We assume τ's deadline is *arbitrary*, i.e., D can be $(<, =, >)$ T. If $D \leq T$, then each job must be finished before the release of the next job, and thus the analysis of τ degrades to the analysis of a single job. When $D > T$, a job may be released before the completion of (and thus be interfered by) previously released jobs, and thus the analysis of a task must look into multiple jobs and quantify the inter-job interference. In this paper, we focus on the case of $D > T$. We assume that an unfinished job *does not* delay future jobs release, and thus more than one job can be executed simultaneously.

B. Schedulability and Capacity Augmentation Bound

The *response time* R_k of a job J_k is $R_k = f_k - r_k$. The *worst-case response time* (WCRT) of task τ is the maximum response time among all its jobs: $R = \max_{\text{all } J_k}\{R_k\}$. A sufficient condition for a task to be schedulable is $R \leq D$.

The *capacity augmentation bound* is a widely used quantitative metric to evaluate performance of real time scheduling algorithms of DAG task:

Definition 1 (**Capacity Augmentation Bound**). *A scheduling algorithm $\mathcal{S}$ has a* capacity augmentation bound β *if any task satisfying the following condition is schedulable by $\mathcal{S}$ on m processors:*

$$\beta \times U \leq m \quad \wedge \quad \beta \times len(G) \leq D$$

The capacity augmentation bound is not only a simple metric for quantifying the worst-case performance of scheduling algorithms but also serves as an efficient schedulability test.

III. Analysis

Our analysis is based on the concept of *lag*, which compares the workload executed by GEDF and a hypothetic ideal scheduler at any time instant. In Section III-A, we establish the relationship between the response time bound and lag. Then in Section III-B, we derive upper bounds for the lag. Putting these results together gives the our schedulability test in Section III-C. Finally, in Section III-D we derive a capacity augmentation bound for GEDF based on the proposed analysis.

A. Response Time and Lag

Without loss of generality, we assume that the system starts from time 0. For any time instant $t > 0$, we use the notation t^- to denote the time $t - \epsilon$ where ϵ is an arbitrary small positive number: $\epsilon \to 0$, and the notation t^+ to denote the time $t + \epsilon$.

Given any two time points $t_1 < t_2$, $A(J_k, t_1, t_2, \mathcal{S})$ denote the total amount of work that job J_k has done under a scheduler $\mathcal{S}$ in time interval $[t_1, t_2)$, and $A(\tau, t_1, t_2, \mathcal{S})$ denotes the sum of $A(J_k, t_1, t_2, \mathcal{S})$ for all jobs generated by τ, i.e.,

$$A(\tau, t_1, t_2, \mathcal{S}) = \sum_{J_k \in \tau} A(J_k, t_1, t_2, \mathcal{S})$$

We know the following by definition:

$$A(\tau, t_1, t_2, \mathcal{S}) = A(\tau, 0, t_2, \mathcal{S}) - A(\tau, 0, t_1, \mathcal{S}) \qquad (1)$$

Since $A(J_k, t_1, t_2, \mathcal{S}) = 0$ for any job J_k that has not been released by time t_2, $A(\tau, t_1, t_2, \mathcal{S})$ only counts for the jobs have been released by t_2.

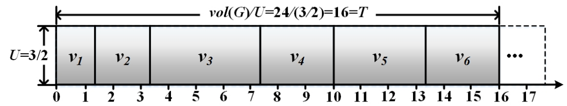


Fig. 2. Illustration of IS.

Let IS denote an ideal scheduler, with which the workload of τ is sequentially executed on a single processor with speed equal to its utilization U. Fig.2 shows a job of task τ in Fig.1 released at time 0 and scheduled by IS, the processor speed is $U = \frac{3}{2}$ and the job is finished within a time interval of length $\frac{vol(G)}{U} = T = 16$. By the definition of IS scheduler, we know:

Lemma 1. *Suppose τ is scheduled by the ideal scheduler IS. For any two time points $t_1 \le t_2$, we have*

$$A(\tau, t_1, t_2, \mathsf{IS}) \le (t_2 - t_1)U \tag{2}$$

Proof. Directly follows the definition of IS. □

Definition 2 (Lag and Maximal Lag). *The lag of jobs of τ at time t under GEDF scheduler is defined as:*

$$lag(\tau, t) = A(\tau, 0, t, \mathsf{IS}) - A(\tau, 0, t, \mathsf{GEDF}) \tag{3}$$

The maximal lag *of task τ under GEDF is defined as:*

$$\mathsf{LAG} = \max_{t>=0}\{lag(\tau, t)\}. \tag{4}$$

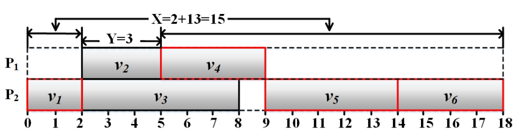


Fig. 3. Illustration of key path.

Definition 3 (Key Path). *The* key path *of job J_k, denoted by $\lambda = \{v_1, v_2, ..., v_p\}$, is a path in G s.t.,*

- *The last vertex v_p in λ has the* latest *finish time among all vertices of job J_k;*
- *$\forall z : 1 < z \le p$, v_{z-1} is a predecessor of v_z with the latest finish time among all predecessors of v_z.*

As shown in Fig.3, a job of task τ is scheduled on two processors whose key path is $\lambda = \{v_1, v_4, v_5, v_6\}$, labeled by the red boxes in the figure.

Lemma 2. *Let $\lambda = \{v_1, v_2, ..., v_p\}$ be a key path of J_k. v_1 is released at r_1 and v_p is finished at f_p. All processors must be busy at any time point in $[r_1, f_p)$ when no vertex on the key path is executing.*

Proof. By Definition 3, $\forall j : 1 \le j \le p-1$, all predecessors of v_{j+1} have been finished by f_j the finish time of v_j, and thus v_{j+1} is eligible from f_j. Therefore, all processors must be busy in $[f_j, s_{j+1})$, where s_{j+1} is the starting time of v_{j+1} (otherwise v_{j+1} will be executed earlier than s_{j+1} since it is eligible and there is an idle processor). □

The following lemma gives the relation between the maximum lag and the response time bounds of a task.

Lemma 3. *Let task τ be scheduled on m processors under GEDF, then the WCRT R of τ is bounded from above by*

$$R \le \frac{\mathsf{LAG} + vol(G) + (m-1)len(G)}{m}.$$

Proof. Prove by contradiction. Let J_k of task τ be scheduled under GEDF, with release time r_k and absolute deadline d_k. Suppose J_k has not finished at time t_k and

$$t_k - r_k > (\mathsf{LAG} + vol(G) + (m-1)len(G))/m. \tag{5}$$

Let $\lambda = \{v_1, v_2, ..., v_p\}$ be a *key path* of J_k and W denote the total amount of work done by J_k during $[r_k, t_k)$. We divide the time interval $[r_k, t_k)$ into two types of subintervals and divide W into two parts accordingly:

- We use X to denote the sum of length of subintervals in which some vertex on the key path λ is executing. Let W_x denote the total amount of work done by J_k during this type of subintervals. We know $W_x \ge X$ since at least one processor is busy during such subintervals (executing vertices on the key path).
- We use Y to denote the sum of length of subintervals in which no vertices on the key path λ are executing. Let W_y denote the total amount of work done by J_k during this type of subintervals. By Lemma 2, all processors are busy in these subintervals, so $W_y = Y \times m$.

By the definition of X and Y we know $X + Y = t_k - r_k$ and $X \le len(\lambda)$ (Fig. 3 illustrates X and Y in $[r_k, t_k) = [0, 18)$). Therefore, we have

$$\begin{aligned} &W \ge X + Y \times m \ge X + (t_k - r_k - X) \times m \\ &= (t_k - r_k) \times m - (m-1)X \ge (t_k - r_k) \times m - (m-1)len(\lambda) \\ &\ge (t_k - r_k) \times m - (m-1)len(G) \end{aligned}$$

and by (5), we have

$$\begin{aligned} W &\ge (\mathsf{LAG} + vol(G) + (m-1)len(G)) - (m-1)len(G) \\ &= \mathsf{LAG} + vol(G). \end{aligned} \tag{6}$$

From the definition of IS we know all jobs released before r_k are finished under IS at r_k. Then we have $lag(\tau, r_k) = A(\tau, 0, r_k, \mathsf{IS}) - A(\tau, 0, r_k, \mathsf{GEDF})$ equals to the unfinished work of jobs released before r_k under GEDF and the unfinished work of jobs pending at r_k (thus with deadlines at or earlier than d_k) is at most $lag(\tau, r_k) + vol(G) \le \mathsf{LAG} + vol(G)$ (note that the work of J_k itself is also included). Since the execution of jobs with deadlines later than d_k does not affect the behavior of jobs with deadlines at or earlier than d_k under GEDF, then by (6), we know the work of all pending jobs at r_k has been finished at time t_k, thus J_k is finished at t_k, reaching a contradiction to our assumption. □

B. Bounding Maximal Lag

In the following we will present *two* upper bounds for the maximum lag of τ. We assume $U \le m$, which is a necessary condition for a task to be schedulable on m processors.

Definition 4 (Busy Interval). *Let task τ be scheduled on m processors, if more than $\lceil U \rceil - 1$ processors are busy during a time interval $I = [t_1, t_2)$, then I is called a* busy interval.

Lemma 4. *For busy interval* $[t_1, t_2)$*,* $lag(\tau, t_1) \geq lag(\tau, t_2)$*.*

Proof. By (1) and (3), we have

$$\begin{aligned}&lag(\tau,t_2)-lag(\tau,t_1)=(A(\tau,0,t_2,\mathsf{IS})-A(\tau,0,t_2,\mathsf{GEDF}))\\&-(A(\tau,0,t_1,\mathsf{IS})-A(\tau,0,t_1,\mathsf{GEDF}))\\=&A(\tau,t_1,t_2,\mathsf{IS})-A(\tau,t_1,t_2,\mathcal{S})\end{aligned}$$

Then by (2) and Definition 4 we have $lag(\tau,t_2)-lag(\tau,t_1) \leq (t_2-t_1)U-(t_2-t_1)\lceil U\rceil \leq 0$. Proved. □

Lemma 4 indicates that the lag of τ does not increase during a busy interval. Based on this observation, we derive the first lag bound in the following lemma.

Lemma 5. *Let task* τ *be scheduled on* m *processors and* $U \leq m$*, then we have:*

$$\mathsf{LAG} \leq vol(G)(\lceil U\rceil - 1).$$

Proof. Consider an arbitrary time instant t_0, and let t_1 denote the last time instant before t_0 when no more than $\lceil U\rceil - 1$ processors are busy at t_1^-. We are guaranteed to find such a time instant since all processors are idle before the system starts running. Since $U \leq m$ and no more than $\lceil U\rceil - 1$ processors are busy at t_1^-, at least one processor is idle at time t_1^-. By GEDF, no processor is allowed to be idle if there are still some jobs ready to be executed, thus at most $\lceil U\rceil - 1$ jobs are pending at time t_1^-. Let $\mathcal{J}$ denote the set of jobs which are pending at time t_1^-, then we have

$$\begin{aligned}lag(\tau,t_1)&=A(\tau,0,t_1,\mathsf{IS})-A(\tau,0,t_1,\mathsf{GEDF})\\&=\sum_{J_k\in\mathcal{J}}(A(J_k,0,t_1,\mathsf{IS})-A(J_k,0,t_1,\mathsf{GEDF}))\\&\leq vol(G)(\lceil U\rceil-1).\end{aligned}$$

Since $[t_1, t_0)$ is a busy interval, then by Lemma 4 we have

$$lag(\tau,t_0)\leq lag(\tau,t_1)\leq vol(G)(\lceil U\rceil-1).$$

Since t_0 is an arbitrarily chosen time instant, we finally have $\mathsf{LAG} \leq vol(G)(\lceil U\rceil - 1)$. □

Then by Lemma 5 we can immediately bound the WCRT:

Corollary 1. *Let* τ *be scheduled on* m *processors under* GEDF *and* $U \leq m$*, we have*

$$R \leq \frac{vol(G)\lceil U\rceil + (m-1)len(G)}{m}.$$

Proof. Proved by combining Lemma 3 and Lemma 5. □

Lemma 5 indicates that the WCRT of a task is bounded even if its utilization U exactly equals to m, i.e., the response time is bounded (i.e., tardiness is bounded in the context of soft real-time scheduling [17]) *without* utilization loss. In the following, we will present the the second lag bound when U is *strictly smaller* than m.

Lemma 6. *Let task* τ *be scheduled on* m *processors and* $U < m$*, then we have:*

$$\mathsf{LAG} \leq \frac{m\times U\times len(G)}{m-U}.$$

Proof. By Lemma 5, we know the LAG of τ is bounded, i.e., it will not forever increase, so there must exist some time instant t_m at which the maximal lag is reached. We assume t_m is such a time instant, i.e., $\mathsf{LAG} = lag(\tau, t_m)$. Therefore, $\forall t \leq t_m$ we have $lag(\tau,t) \leq lag(\tau,t_m)$, so

$$lag(\tau,t_m)-lag(\tau,t)=A(\tau,t,t_m,\mathsf{IS})–A(\tau,t,t_m,\mathsf{GEDF})\geq 0,$$

and by (2), we have

$$A(\tau,t,t_m,\mathsf{GEDF}) \leq (t_m-t)U. \tag{7}$$

Let $\mathcal{J}$ denote the set of jobs pending at t_m and J_p denotes the earliest released job in $\mathcal{J}$ with release time of r_p. Clearly $r_p \leq t_m$, then by (7) we have $A(\tau,r_p,t_m,\mathsf{GEDF}) \leq (t_m - r_p)U$. Let $\mathcal{L}$ denote the cumulative length of subintervals where all processors are busy during $[r_p, t_m)$, then it must satisfy:

$$\mathcal{L} \leq \frac{A(\tau,r_p,t_m,\mathsf{GEDF})}{m} \leq \frac{(t_m-r_p)U}{m}. \tag{8}$$

Let $\mathcal{Z}$ denote the cumulative length of subintervals in $[r_p, t_m)$ where not all processors are busy. Let $\lambda = \{v_1, v_2, ..., v_k\}$ denote a key path of J_p. Then by Lemma 2, we know some vertex on the key path must be executing during subintervals in which some processor is idle in $[r_p, t_m)$. Since J_p is still pending at time t_m, it must be true that $\mathcal{Z} < len(\lambda)$ (since otherwise J_p has finished at time t_m). Then by combining with (8) and $\mathcal{L} + \mathcal{Z} = t_m - r_p$ we have

$$\begin{aligned}&t_m-r_p<(t_m-r_p)U/m+len(\lambda)\\\Rightarrow\quad&t_m-r_p<(t_m-r_p)U/m+len(G)\\\Rightarrow\quad&t_m-r_p<m\times len(G)/(m-U).\end{aligned}\tag{9}$$

Since J_p is the first pending job at time t_m, all jobs released before J_p have finished at t_m. Therefore we have

$$\begin{aligned}lag(\tau,t_m)&=\sum_{J_k\in\mathcal{J}}(A(J_k,0,t_m,\mathsf{IS})-A(J_k,0,t_m,\mathsf{GEDF}))\\&\leq A(\tau,r_p,t_m,\mathsf{IS})\end{aligned}$$

and by (2), we have $lag(\tau,t_m) \leq (t_m-r_p)U$. By combining this with (9) the lemma is proved. □

The above lag bound gives a WCRT bound:

Corollary 2. *Let* τ *be scheduled on* m *processors by GEDF where* $U < m$*, we have*

$$R \leq \frac{U\times len(G)}{m-U} + \frac{vol(G)+(m-1)len(G)}{m}.$$

Proof. Proved by combining Lemma 3 and Lemma 6. □

C. Schedulability Test

A sufficient condition for a task τ to be schedulable is $R \leq D$, then by Corollary 1 and 2 we have:

Theorem 1. *Any DAG task* τ *is schedulable on* m *processors when* $u \leq m$ *if at least one of the following conditions holds:*

$$\frac{vol(G)\lceil U\rceil + (m-1)len(G)}{m} \leq D \tag{10}$$

$$\frac{U\times len(G)}{m-U} + \frac{vol(G)+(m-1)len(G)}{m} \leq D. \tag{11}$$

Conditions (10) and (11) are in general incomparable with each other. Therefore, we combine them to achieve better analysis accuracy. However, in Section IV we evaluate them separately to explore their strength with different parameters.

D. Capacity Augmentation Bound

The best known capacity augmentation bound for our considered problem is 2.5 [10]. In the following we improve it to $\sqrt{2}+1 \approx 2.41$ based on the above schedulability test.

Theorem 2. GEDF *has a capacity augmentation bound of* $\sqrt{2}+1$ *for DAG task* τ *with* $D > T$ *on* m *processors.*

Proof. We need to prove that τ is schedulable if both the following conditions hold:

$$(\sqrt{2}+1)len(G) \leq D \tag{12}$$

$$(\sqrt{2}+1)U \leq m. \tag{13}$$

We consider two cases:

Case 1: $\frac{vol(G)+(m-1)len(G)}{m} \leq T$. From [12], we know

$$R \leq \frac{vol(G)+(m-1)len(G)}{m},$$

so we know $R \leq T$, and since $T < D$, τ is schedulable.

Case 2: $\frac{vol(G)+(m-1)len(G)}{m} > T$. Combining this condition with (12) and (13) gives:

$$\begin{aligned} & T < \frac{(\sqrt{2}-1)mT + (m-1)(\sqrt{2}-1)D}{m} \\ \Rightarrow\quad & T < \frac{(m-1)(\sqrt{2}-1)D}{(2-\sqrt{2})m} \end{aligned} \tag{14}$$

Then by combining (14) and (13), we have

$$vol(G) \leq \frac{mT}{\sqrt{2}+1} \leq \frac{(m-1)(\sqrt{2}-1)^2 D}{(2-\sqrt{2})}$$

$$\begin{aligned} \Rightarrow\quad & \frac{U \times len(G)}{m-U} + \frac{vol(G)+(m-1)len(G)}{m} \\ & \leq \frac{U \times len(G)}{m-U} + \frac{vol(G)+(m-1)(\sqrt{2}-1)D}{m} \\ & \leq \frac{(\sqrt{2}-1)D}{\sqrt{2}} + \frac{(m-1)(\sqrt{2}-1)D}{(2-\sqrt{2})m} \\ & = D - \frac{\sqrt{2}}{2}D + \frac{\sqrt{2}}{2}D - \frac{D}{\sqrt{2}m} < D \end{aligned}$$

By Theorem 1 (condition (11)), we know τ is schedulable. □

E. Extention to Multiple DAG Tasks

The schedulability tests in Theorem 1 can be easily extended to the analysis of multiple DAG tasks under *federated scheduling* [4], where each heavy task ($U > 1$) is assigned several dedicated processors and exclusively executes on them while all light tasks ($U \leq 1$) share the remaining processors together as if they are sequential tasks.

Our schedulability test for a DAG task set has two parts: (i) assign dedicated processors to each heavy task according to Theorem 1 if there are enough processors, otherwise the test returns *failure*, and (ii) test whether all light tasks are schedulable on the remaining processors. The pseudo-codes are shown in Algorithm 1, where the function Sched(S,Ω) returns the schedulability test result of light tasks on $|\Omega|$ processors by algorithm PARTITION in [7] (for arbitrary-deadline sequential tasks under partitioned EDF [18]).

If Algorithm 1 returns *success*, then at runtime we schedule each heavy task exclusively on its dedicated processors by GEDF and light tasks on other processors by partitioned EDF.

Algorithm 1 Schedulability test under federated scheduling.

for each heavy task τ **do**
    find the minimal value m to satisfy (10) or (11).
    if less than m processors are available **then**
        return *failure*
    end if
    assign m dedicated processors to τ
end for
$\Omega =$ the set of remaining processors
$S =$ the set of light tasks
if Sched(S, Ω) **then return** *success* **else return** *failure*

IV. Evaluations

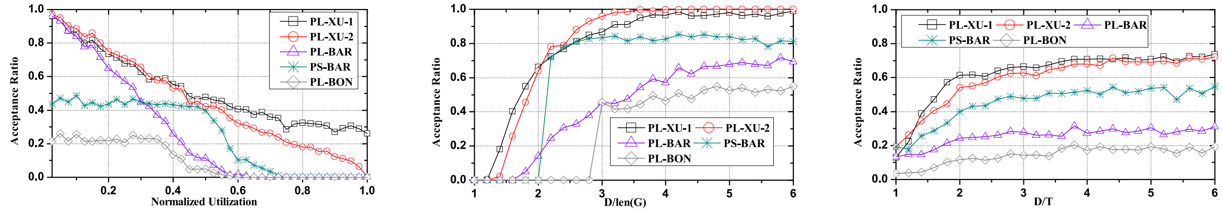
We evaluate our analysis techniques with both the single- and multiple-task settings. The tasks are generated as follows:

- **Task Graph** $G = \langle V, E \rangle$**:** The graph is generated using the Erdös-Rényi method $G(n_i, p)$ [19]. For each task, the number of vertices is randomly chosen in $[50, 250]$. The WCET of each vertex is randomly picked in $[50, 100]$. For each possible edge we generate a random value in $[0, 1]$ and add the edge to the graph only if the generated value is less than a predefined threshold $p = 0.1$. The same as in [13], a minimum number of additional edges are added to make a task graph weakly connected.
- **Deadline and Period:** The deadline is generated as an integer of powers of two. We find the smallest value a such that $len(G) \leq 2^a$, and randomly set D to be one of 2^a, 2^{a+1}, or 2^{a+2}. The ratio $len(G)/D$ of the task is in the range (1/2, 1], (1/4,1/2], or (1/8, 1/4], when its deadline D is 2^a, 2^{a+1}, or 2^{a+2}, respectively. The period T is uniformly selected from the range $[0.1 \times D, D]$.

For each task set, we generate n tasks where n is randomly chosen from $[2, 16]$ (n is set to be 1 in Section IV-A for the case of a single task), and the normalized utilization U_{norm} (the ratio between the total utilization of the task set and the number of processors) is predefined, which will be explained in detail for results in each figure in the following. After we generate all tasks in a task set, we can compute the total utilization $U_{\sum}$, then we set the number of processors according to the formula $m = \lceil \frac{U_{\Sigma}}{U_{norm}} \rceil$. For each configuration (corresponding to one point on the X-axis), we generate 1000 task sets.

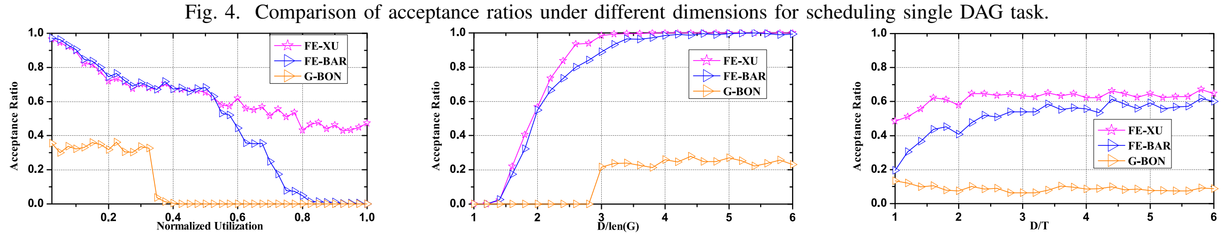
A. Single Task

We evaluate the two test conditions (10) and (11) in Theorem 1 separately, and denote them as PL-XU-1 and PL-XU-2 respectively. We compare the acceptance ratios (i.e., the



(a) Under different normalized utilizations (b) Under different $D/len(G)$ (c) Under different D/T

Fig. 4. Comparison of acceptance ratios under different dimensions for scheduling single DAG task.



(a) Under different normalized utilizations (b) Under different $D/len(G)$ (c) Under different D/T

Fig. 5. Comparison of acceptance ratios under different dimensions for scheduling multiple DAG tasks.

ratio between the number of schedulable task sets and the number of all tested task sets) with the state-of-the-art analysis techniques: (i) condition (2) in [10], denoted by PL-BAR, (ii) the algorithm in [10], denoted by PS-BAR, and (iii) Theorem 21 in [3], denoted by PL-BON (which is for multiple tasks, but also applicable for a single task). In these notations, the prefix "PL" means the test is of polynomial time complexity and the prefix "PS" means pseudo-polynomial time complexity.

Fig.4.(a) shows acceptance ratios of all tests under different normalized utilizations (X-axis). The result shows that both of our two tests perform better than others. The configurations in Fig.4.(b) are the same as in Fig.4.(a). The only difference is that deadlines are generated according to different $D/len(G)$ (X-axis) and U_{norm} is randomly chosen from [0.3, 0.8]. We can observe that PL-XU-2 performs better than PL-XU-1 for larger $D/len(G)$. In Fig.4.(c), periods are generated according to different D/T (X-axis) and U_{norm} is randomly chosen from [0.3, 0.8]. The acceptance ratios of all tests increase as D/T becomes larger and our two tests also outperform others.

The above results show consistent better performance of our tests than the state-of-the-art under different parameter settings. Our two tests do not dominate each other. PL-XU-1 can schedule some tasks with high U_{norm} and PL-XU-2 performs better with larger $D/len(G)$.

B. Multiple Tasks

We compare the acceptance ratio of Algorithm 1 denoted by FE-XU for scheduling multiple tasks with the state-of-the-art: (i) Algorithm FEDERATED in [7] also using the federated scheduling approach, denoted by FE-BAR, and (ii) the schedulability test for GEDF in [3] (Theorem 21), denoted by G-BON. The configuration of each figure in Fig.5 is the same as its correspondence in Fig.4. The trends of all tests in Fig.5 are the same as in Fig.4, and our test outperforms the state-of-the-art for scheduling multiple DAG tasks.

V. CONCLUSIONS

In this paper, we develop new analysis techniques for scheduling a DAG task with arbitrary deadline on a multi-cores platform under GEDF, for which we prove a capacity augmentation bound of $\sqrt{2}+1$. The techniques are extended to schedule multiple DAG tasks under federated scheduling. The experimental evaluation results show that our test consistently outperforms all others in different dimensions.

ACKNOWLEDGMENT

This work is supported by the Research Grants Council of Hong Kong (GRF 15204917 and 15213818).

REFERENCES

[1] A. OpenMP, "Openmp application program interface version 4.0," 2013.
[2] J. Li, K. Agrawal, and et.al, "Analysis of global edf for parallel tasks," in *ECRTS*, 2013.
[3] V. Bonifaci, A. Marchetti-Spaccamela, and et.al, "Feasibility analysis in the sporadic dag task model," in *ECRTS*, 2013.
[4] J. Li, J. J. Chen, and et.al, "Analysis of federated and global scheduling for parallel real-time tasks," in *ECRTS*, 2014.
[5] A. Saifullah, J. Li, and K. Agrawal, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems*, 2013.
[6] X. Jiang, X. Long, and et.al, "On the decomposition-based global edf scheduling of parallel real-time tasks," in *RTSS*, 2016.
[7] S. Baruah, "Federated scheduling of sporadic dag task systems," in *IPDPS*, 2015.
[8] ——, "The federated scheduling of constrained-deadline sporadic dag task systems," in *DATE*, 2015.
[9] ——, "The federated scheduling of systems of conditional sporadic dag tasks," in *EMSOFT*, 2015.
[10] S. Baruah, V. Bonifaci, and et.al, "A generalized parallel task model for recurrent real-time processes," in *RTSS*, 2012.
[11] M. R. Garey, "A guide to the theory of np-completeness," *Computers and intractability*, 1979.
[12] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM journal on Applied Mathematics*, 1969.
[13] A. Saifullah, D. Ferry, and et.al, "Parallel real-time scheduling of dags," *Parallel and Distributed Systems, IEEE Transactions on*, 2014.
[14] M. Qamhieh, F. Fauberteau, and et.al, "Global edf scheduling of directed acyclic graphs on multiprocessor systems," in *RTNS*, 2013.
[15] M. Qamhieh, L. George, and S. Midonnet, "A stretching algorithm for parallel real-time dag tasks on multiprocessor systems," in *RTNS*, 2014.
[16] J. Xu, G. Nan, and et.al, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *RTSS*, 2017.
[17] U. C. Devi and J. H. Anderson, "Tardiness bounds under global edf scheduling on a multiprocessor," in *RTSS*, 2005.
[18] S. K. Baruah and N. W. Fisher, "The partitioned dynamic-priority scheduling of sporadic task systems," *Real-Time Systems*, 2007.
[19] D. Cordeiro, G. Mounié, and et.al, "Random graph generation for scheduling simulations," in *ICST*, 2010.