2013 25th Euromicro Conference on Real-Time Systems

Analysis of Global EDF for Parallel Tasks

Jing Li, Kunal Agrawal, Chenyang Lu, Christopher Gill
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO, USA
{li.jing, kunal, lu, and cdgill}@wustl.edu

***Abstract*—As multicore processors become ever more prevalent, it is important for real-time programs to take advantage of intra-task parallelism in order to support computation-intensive applications with tight deadlines. We prove that a *Global Earliest Deadline First (GEDF)* scheduling policy provides a *capacity augmentation bound of* $4-\frac{2}{m}$ and a *resource augmentation bound of* $2-\frac{1}{m}$ for parallel tasks in the general *directed acyclic graph model*. For the proposed *capacity augmentation bound* of $4-\frac{2}{m}$ for implicit deadline tasks under GEDF, we prove that if a task set has a total utilization of at most $m/(4-\frac{2}{m})$ and each task's critical path length is no more than $1/(4-\frac{2}{m})$ of its deadline, it can be scheduled on a machine with m processors under GEDF. Our capacity augmentation bound therefore can be used as a straightforward schedulability test. For the standard *resource augmentation bound* of $2-\frac{1}{m}$ for arbitrary deadline tasks under GEDF, we prove that if an ideal optimal scheduler can schedule a task set on m unit-speed processors, then GEDF can schedule the same task set on m processors of speed $2-\frac{1}{m}$. However, this bound does not lead to a schedulabilty test since the ideal optimal scheduler is only hypothetical and is not known. Simulations confirm that the GEDF is not only safe under the capacity augmentation bound for various randomly generated task sets, but also performs surprisingly well and usually outperforms an existing scheduling technique that involves task decomposition.**

***Index Terms*—real-time scheduling, parallel scheduling, global EDF, resource augmentation bound, capacity augmentation bound**

I. Introduction

During the last decade, the performance increase of processor chips has come primarily from increasing numbers of cores. This has led to extensive work on real-time scheduling techniques that can exploit multicore and multiprocessor systems. Most prior work has concentrated on ***inter-task*** parallelism, where each task runs sequentially (and therefore can only run on a single core) and multiple cores are exploited by increasing the number of tasks. This type of scheduling is called ***multiprocessor scheduling***. When a model is limited to inter-task parallelism, each individual task's total execution requirement must be smaller than its deadline since individual tasks cannot run any faster than on a single-core machine. In order to enable tasks with higher execution demands and tighter deadlines, such as those used in autonomous vehicles, video surveillance, computer vision, radar tracking and real-time hybrid testing [1], we must enable parallelism within tasks.

In this paper, we are interested in parallel scheduling, where in addition to inter-task parallelism tasks sets contain ***intra-task*** parallelism, which allows threads from one task running in parallel more than single core. There has been some recent work in this area. Many of these approaches are based on task decomposition [2]–[4], which first decomposes each parallel task into a set of sequential subtasks with assigned intermediate release times and deadlines, and then schedules these sequential subtasks using a known multiprocessor scheduling algorithm. Decomposition techniques require, in addition to task-level worst case execution time and critical path length, also a thorough knowledge of the structure of tasks as well as the individual worst case execution time of each subtask prior to execution. Such knowledge is expensive to acquire and may be inaccurate, pessimistic or even unavailable when tasks from different vendors are integrated on a common computing platform. Moreover, decomposition introduces implementation complexities in real systems [5].

Therefore, we are interested in analyzing the performance of a ***global EDF*** (***GEDF***) scheduler *without* any decomposition. We consider a general task model, where each task is represented as a ***directed acyclic graph (DAG)*** where each node represents a sequence of instructions (thread) and each edge represents a dependency between nodes. A node is ***ready*** to be executed when all its predecessors have been executed. GEDF works as follows: at each time step, the scheduler first tries to schedule as many ready nodes from all jobs with the earliest deadline as it can; then it schedules ready nodes from the jobs with the next earliest deadline, and so on, until either all processors are busy or no more nodes are ready.

Compared with other schedulers, GEDF has benefits, such as automatic load balancing. Efficient and scalable implementations of GEDF for sequencial tasks are available for Linux [6] and LITMUS [7], suggesting the potential existence of an easy implementation for parallel tasks if decomposition is not required. Prior theory work analyzing GEDF for parallel tasks is either restricted to a single recurring task [8] or considers response time analysis for soft-real time tasks [9]. In this work, we consider task sets with n tasks and analyze their schedulability under the GEDF scheduler in terms of augmentation bounds.

We distinguish between two types of augmentation bounds, both of which are called "resource augmentation" in previous literature. By standard definition, a scheduler $\mathcal{S}$ provides a ***resource augmentation bound*** of b if the following condition holds: if an ideal scheduler can schedule a task set on m unit-speed processors, then $\mathcal{S}$ can schedule that task set on m processors of speed b. Note that the ***ideal scheduler (optimal***

1068-3070/13 $26.00 © 2013 IEEE
DOI 10.1109/ECRTS.2013.12

schedule) is only a hypothetical scheduler, meaning that if a feasible schedule ever exists for a task set then this ideal scheduler can guarantee to schedule it. Unfortunately, Fisher et al. [10] proved that optimal online multiprocessor scheduling of sporadic task systems is impossible. Since there may be no way to tell whether the ideal scheduler can schedule a given task set on unit-speed processors, a resource augmentation bound may not provide a schedulability test.

Therefore, we distinguish resource augmentation from a ***capacity augmentation bound*** which can serve as an easy schedulability test. If on unit-speed processors, a task set has total utilization of at most m and the critical path length of each task is smaller than its deadline, then scheduler $\mathcal{S}$ with capacity augmentation bound of b can schedule this task set on m processors of speed b. Capacity augmentation bounds have the advantage that they directly lead to schedulability tests, since one can easily check the bounds on utilization and critical path length for any task set.

The contributions presented in this paper are as follows:

- For a system with m identical processors, we prove a capacity augmentation bound of $4-\frac{2}{m}$ (which approaches to 4 as m approaches to infinity) for sporadic task sets with *implicit deadlines* — the relative deadline of each task is equal to its period. Another way to understand this bound is: if a task set has total utilization at most $m/(4-\frac{2}{m})$ and the critical path length of each task is at most $1/(4-\frac{2}{m})$ of its deadline, then it can be scheduled using GEDF on unit-speed processors.
- For a system with m identical processors, we prove a resource augmentation bound of $2-\frac{1}{m}$ (which approaches to 2 as m approaches to infinity) for sporadic task sets with *arbitrary deadlines*.
- We also show that GEDF's capacity bound for parallel task sets (even with implicit deadlines) is greater than $2-\frac{1}{m}$. In particular, we show example task sets with utilization m where the critical path length of each task is no more than its deadline, while GEDF misses a deadline on m processors with speed less than $\frac{3+\sqrt{5}}{2} \approx 2.618$.
- We conduct simulation experiments to show that the capacity augmentation bound is safe for task sets with different DAG structures (as mentioned above, checking the resource augmentation bound is difficult since we cannot compute the optimal schedule). Results show that GEDF performs surprisingly well. All simulated random task sets meet their deadlines with processor speed of 2. We also compare GEDF with a scheduling technique that decomposes parallel tasks and then schedules decomposed subtasks using GEDF [4]. We find that for most task sets, GEDF is better without decomposition. Only among task sets, which are intentionally designed to be harder for GEDF to schedule, does the decomposition technique occasionally perform better than GEDF without decomposition.

Section II reviews related work. Section III describes the DAG task model with intra-task parallelism. Proofs for a capacity augmentation bound of $4-\frac{2}{m}$ and a resource augmentation bound of $2-\frac{1}{m}$ are presented in Sections IV and V respectively. Section VI shows experimental results and Section VII gives concluding remarks.

II. Related Work

Scheduling parallel tasks without deadlines has been addressed by parallel-computing researchers [11]–[14]. ***Soft real-time scheduling*** has been studied for various optimization criteria, such as cache misses [15], [16], makespan [17] and total work done by tasks that meet deadlines [18]. Most prior work on ***hard real-time scheduling*** atop multiprocessors has concentrated on sequential tasks [19]. In this context, many sufficient schedulability tests for GEDF and other global fixed priority scheduling algorithms have been proposed [20]–[28].

Earlier work considering intra-task parallelism makes strong assumptions on task models [29] [30] [31]. For more realistic parallel tasks, ***synchronous tasks***, Kato et al. [32] proposed a gang scheduling approach. The synchronous model, a special case of the DAG model, represents tasks with a sequence of multithreaded segments with synchronization points between them (such as those generated by parallel-for loops).

Most other approaches for scheduling synchronous tasks involve decomposing parallel tasks into independent sequential subtasks, which are then scheduled using known multiprocessor scheduling techniques, such as [24], [33]. For a restricted set of synchronous tasks, Lakshmanan et al. [2] prove a capacity augmentation bound of 3.42 using deadline monotonic scheduling for decomposed tasks. For more general synchronous tasks, Saifullah et al. [3] proved a capacity augmentation bound of 4 for GEDF and 5 for deadline monotonic scheduling. This decomposition approach was recently extended to general DAGs [4] to achieve a capacity augmentation bound of 4 under GEDF on decomposed tasks (note that in that work GEDF is used to schedule sequential decomposed tasks, not parallel tasks directly). This is the best augmentation bound known for task sets with multiple DAGs.

More recently, there has been some work on scheduling general DAGs without decomposition. Nogueira et al. [34] explored the use of work-stealing for real-time scheduling. The paper is mostly experimental and focused on soft real-time preformance. The bounds for hard real-time scheduling only guarantee that tasks meet deadlines if their utilization is smaller than 1. This restriction limits its application to many computation-intensive real-time applications, whose utilization exceeds the capacity of a single core. For example, even for large m, like 101, a task with utilization $m/100$ is not guaranteed to be schedulable using the such schedulers. Anderson et al. [9] analyzed the response time of GEDF without decomposition for soft real-time tasks. Baruah et al. [8] proved when the task set is a *single DAG task* with arbitrary deadlines, GEDF provides a resource augmentation bound of 2. In this paper, instead of just a single task, we consider multiple DAG tasks with different execution times, release times and deadlines. For the resource augmentation bound of $2-\frac{1}{m}$ for multiple DAGs under GEDF proved in our paper, Bonifaci

et al. [35] also show the same resource augmentation bound $2 - \frac{1}{m}$. Our work was done independently of that work, and they also do not consider capacity augmentation.

III. Task Model

This section presents a model for DAG tasks. We consider a system with m identical unit-speed processors. The task set τ consists of n tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task τ_i is represented by a directed acyclic graph (DAG), and has a period P_i and deadline D_i. We represent the j-th subtask of the ith task as node W_i^j. A directed edge from node W_i^j to W_i^k means that W_i^k can only be executed after W_i^j has finished executing. A node is ***ready*** to be executed as soon as all of its predecessors have been executed. Each node has its own worst case execution time C_i^j. Multiple source nodes and sink nodes are allowed in the DAG, and the DAG is not required to be fully connected. Figure 1 shows an example of a task consisting of 5 subtasks in the DAG structure.

For each task τ_i in task set τ, let $C_i = \sum_j C_i^j$ be the total worst case execution time on a single processor, also called the ***work*** of the task. Let L_i be the critical path length (i.e. the worst case execution time of the task on infinite number of processors). In Figure 1, the critical path (i.e. the longest path) starts from node W_1^1, goes through W_1^3 and ends at node W_1^4, so the critical path length of DAG W_1 is $1 + 3 + 2 = 6$. The work and the critical path length of any job generated by task τ_i are the same as those of task τ_i.

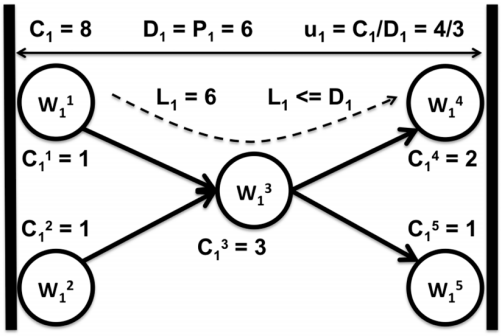

Fig. 1: Example task with work $C_i = 8$ and critical path length $L_i = 6$.

We also define the notion of ***remaining work*** and ***remaining critical path length*** of a partially executed job. The remaining work is the total work minus the work that has already been done. The remaining critical path length is the length of the longest path in the unexecuted portion of the DAG (including partially executed nodes). For example, in Figure 1, if W_1^1 and W_1^2 are completely executed, and W_1^3 is partially executed such that 1 (out of 3) units of work has been done for it, then the remaining critical path length is $2 + 2 = 4$.

Nodes do not have individual release offsets and deadlines when scheduled by the GEDF scheduler; they share the same absolute deadline of their jobs. Therefore, to analyze the GEDF scheduler, we do not require any knowledge of the DAG structure beyond the total worst case execution time C_i, deadline D_i, period P_i and critical path length L_i. We also define the utilization of a task τ_i as $u_i = \frac{C_i}{P_i}$.

On unit speed processors, a task set is not schedulable (by any scheduler) unless the following conditions hold:

- The critical path length of each task is less than its deadline.
$$L_i \leq D_i \tag{1}$$
- The total utilization is smaller than the number of cores.
$$\sum_i u_i \leq m \tag{2}$$

In addition, we denote $J_{k,a}$ as the a-th job instance of task k in system execution. For example, the i-th node of $J_{k,a}$ is represented as $W_{k,a}^i$. We denote $r_{k,a}$ and $d_{k,a}$ as the absolute release time and absolute deadline of job $J_{k,a}$ respectively. Relative deadline D_k is equal to $d_{k,a} - r_{k,a}$. Since in this paper we address sporadic tasks, the absolute release time has the following properties:

$$\begin{aligned} r_{k,a+1} &\geq d_{k,a} \\ r_{k,a+1} - r_{k,a} &\geq d_{k,a} - r_{k,a} = D_k \end{aligned}$$

IV. Capacity Augmentation Bound of $4 - \frac{2}{m}$

In this section, we propose a capacity augmentation bound of $4 - \frac{2}{m}$ for *implicit deadline tasks*, which yields an easy schedulability test. In particular, we show that GEDF can successfully schedule a task set, if the task set satisfies the following conditions: (1) its total utilization is at most $m/(4 - \frac{2}{m})$ and (2) the critical path length of each task is at most $1/(4 - \frac{2}{m})$ of its period (and deadline). Note that this is equivalent to say that if a task set meets conditions from Inequalities 1 and 2 of processors of unit speed, then it can be scheduled on m processors of speed $4 - \frac{2}{m}$ (which approaches to 4 as m approaches to infinity).

The gist of the proof is the following: at a job's release time, we can bound the remaining work from other tasks under GEDF with speed up $4 - \frac{2}{m}$. Bounded remaining work leads to bounded interference from other tasks, and hence GEDF can successfully schedule all of them.

Notation

We first define a notion of interference. Consider a job $J_{k,a}$, which is the a-th instance of task τ_k. Under GEDF scheduling, only jobs that have absolute deadlines earlier than the absolute deadline of $J_{k,a}$ can interfere with $J_{k,a}$. We say that a job is ***unfinished*** if the job has been released but has not completed yet. Due to implicit deadlines ($D_i = P_i$), at most one job of each task can be unfinished at any time.

There are two sources of interference for job $J_{k,a}$. (1) ***Carry-in work*** is the work from jobs that were released before $J_{k,a}$, did not finish before $J_{k,a}$ was released, and have deadlines before $J_{k,a}$ deadline. Let $R_i^{k,a}$ be the carry-in work due to task i and let $R^{k,a} = \sum_i R_i^{k,a}$ be the total carry-in from the entire task set onto the job $J_{k,a}$. (2) Other than carry-in work, the jobs that were released after (or at the same time as) $J_{k,a}$ was released can also interfere with it if their deadlines are either before or at the same time as $J_{k,a}$. Let $n_i^{k,a}$ be the number of task i's jobs, which are released after the release time of $J_{k,a}$ but have deadlines no later than the deadline of

$J_{k,a}$ (that is, the number of jobs from task i that entirely fall in between the release time and deadline of $J_{k,a}$, i.e. the time interval $[r_{k,a}, d_{k,a}]$.) For example, in right part of Figure 2, one entire job $J_{1,3}$ falls within time interval $[r_{3,1}, d_{3,1}]$ of job $J_{3,1}$. So, $n_1^{3,1} = 1$. By definition and $D_i = P_i$, every task i has the property that

$$n_i^{k,a} D_i \leq D_k \tag{3}$$

Therefore, the total amount of work, $A^{k,a}$ that can interfere with $J_{k,a}$ (including $J_{k,a}$'s work) and must be finished (to prevent any deadline misses) before the deadline of $J_{k,a}$ is the sum of the carry-in work and the work that was released at or after $J_{k,a}$'s release.

$$A^{k,a} = R^{k,a} + \sum_i u_i n_i^{k,a} D_i. \tag{4}$$

Note that the work of the job $J_{k,a}$ itself is also included in this formula. That is, in this formulation, each job interferes with itself.

Proof of the Theorem

Consider a GEDF schedule with m processors each of speed b. Each time step can be divided into b sub-steps such that each processor can do one unit of work in each sub-step. We say a sub-step is ***complete*** if all processors are working during that sub-step, otherwise, we say it is ***incomplete***.

First, a couple of straight-forward lemmas.

Lemma 1. *On every incomplete sub-step, the remaining critical path length of each unfinished job reduces by 1.*

Lemma 2. *In any t contiguous time steps (bt sub-steps) with unfinished jobs, if there are t^* incomplete sub-steps, then the total work done during this time, F_t is at least*

$$F^t \geq bmt - (m-1)t^*.$$

Proof: The total number of complete sub-steps during t steps is $bt - t^*$, and the total work during these complete steps is $m(bt - t^*)$. On an incomplete sub-step, at least one unit of work is done. Therefore, the total work done in incomplete sub-steps is at least t^*. Adding the two gives us the bound. ■

We now prove a sufficient condition for the schedulability of a particular job.

Lemma 3. *If interference $A^{k,a}$ on a job $J_{k,a}$ is bounded by*

$$A^{k,a} \leq bmD_k - (m-1)D_k,$$

then job $J_{k,a}$ can meet its deadline on m identical processors with speed of b.

Proof: Note that there are D_k time steps (therefore bD_k sub-steps) between the release time and deadline of this job. There are two cases:

Case 1: The total number of incomplete sub-steps between the release time and deadline of $J_{k,a}$ is more than D_k, and therefore, also more than L_k. In this case, $J_{k,a}$'s critical path length reduces on all of these sub-steps. After at most L_k incomplete steps, the critical path is 0 and the job has finished executing. Therefore, it can not miss the deadline.

Case 2: The total number of incomplete sub-steps between the release and deadline of $J_{k,a}$ is smaller than D_k. Therefore, the total amount of work done during this time is more than $bmD_k - (m-1)D_k$ by the condition in Lemma 2. Since the total interference (including $J_{k,a}$'s work) is at most this quantity, the job cannot miss its deadline. ■

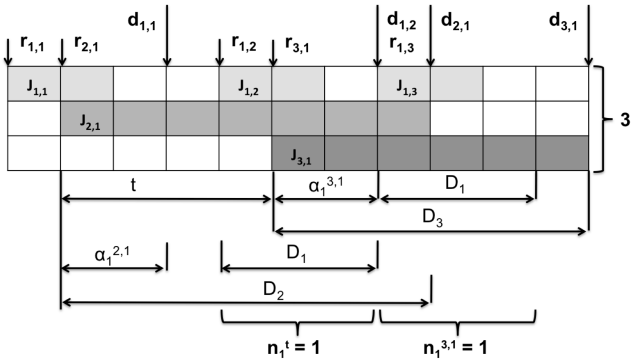

Fig. 2: Example of tasks execution trace

We now define additional notation in order to prove that if the carry-in work for a job is bounded, then GEDF guarantees a capacity augmentation bound of b. Let $\alpha_i^{k,a}$ be the number of time steps between the absolute release time of $J_{k,a}$ and the absolute deadline of the carry-in job of task i. Hence, for $J_{k,a}$ and its carry-in job $J_{j,b}$ of task j

$$\alpha_j^{k,a} = d_{j,b} - r_{k,a} \tag{5}$$

To make the notation clearer, we give an example in Figure 2. There are 3 sporadic tasks with implicit deadlines: the (execution time, deadline, period) for task 1, 2 and 3 are (2, 3, 3), (7, 7, 7) and (6, 6, 6) respectively. For simplicity, assume they are sequential tasks. Since tasks are sporadic, $r_{1,2} > d_{1,1}$. $\alpha_1^{3,1}$ is the number of time steps between the release time of job $J_{3,1}$ and the deadline of the carry-in job $J_{1,2}$ from task 1. In this example, $\alpha_1^{3,1} = 2$. Similarly, $\alpha_2^{3,1} = 3$. Also, $n_1^{3,1} = 1$.

For either periodic or sporadic tasks, task i has the property

$$\alpha_i^{k,a} + n_i^{k,a} D_i \leq D_k \tag{6}$$

since $\alpha_i^{k,a}$ is the remaining length of the carry-in job and $n_i^{k,a}$ is the number of jobs of task i entirely falling in the period (relative deadline) of job $J_{k,a}$. As in Figure 2, $\alpha_1^{3,1} + n_1^{3,1} D_1 = 2 + 1 * 3 = 5 < 6 = D_3$.

Lemma 4. *If the processors' speed is $b \geq 4 - \frac{2}{m}$ and the total carry-in work $R^{k,a}$ from every task i satisfies the condition*

$$R^{k,a} \leq \sum_i u_i \alpha_i^{k,a} + m \cdot \max_i (\alpha_i^{k,a}),$$

then job $J_{k,a}$ always meets its deadline under global EDF.

Proof: The total amount of interfering work (including $J_{k,a}$'s work) is $A^{k,a} = R^{k,a} + \sum_i u_i n_i^{k,a} D_i$. Hence, according to the condition in Lemma 4, the total amount of work

$$\begin{aligned} A^{k,a} &= R^{k,a} + \sum_i u_i n_i^{k,a} D_i \\ &\leq \sum_i u_i \alpha_i^{k,a} + m \max_i (\alpha_i^{k,a}) + \sum_i u_i n_i^{k,a} D_i \\ &\leq \sum_i u_i (\alpha_i^{k,a} + n_i^{k,a} D_i) + m \max_i (\alpha_i^{k,a}) \end{aligned}$$

Using eq.(6) to substitute D_k into the formula, then

$$A^{k,a} \leq \sum_i u_i D_k + m D_k$$

Since the total task set utilization does not exceed the number of processors m, by eq.(2), we replace $\sum_i u_i$ with m. And since $b \geq 4 - \frac{2}{m}$ and $m \geq 1$, we get

$$\begin{aligned} A^{k,a} &\leq 2mD_k \leq (3m-1)D_k \\ &\leq (4 - \frac{2}{m})mD_k - (m-1)D_k \\ &\leq bmD_k - (m-1)D_k \end{aligned}$$

Finally, according to Lemma 3, since the interference satisfies the bound, job $J_{k,a}$ can meet its deadline. ■

We now complete the proof by showing that the carry-in work is bounded as required by Lemma 4 for every job.

Lemma 5. *If the processor's speed $b \geq 4 - \frac{2}{m}$, then, for either periodic or sporadic task sets with implicit deadlines, the total carry-in work $R^{k,a}$ for every job $J_{k,a}$ in the task set is bounded by*

$$R^{k,a} \leq \sum_i u_i \alpha_i^{k,a} + m \max_i (\alpha_i^{k,a})$$

Proof: We prove this theorem by induction from absolute time 0 to the release time of job $J_{k,a}$.

Base Case: For the very first job of all the tasks released in the system (denoted $J_{l,1}$), no carry-in jobs are released before this job. Therefore, the condition trivially holds and the job can meet its deadline by Lemma 4.

$$R^{l,1} = 0 \leq \sum_i u_i \alpha_i^{l,1} + m \max_i (\alpha_i^{l,1})$$

Inductive Step: Assume that for every job with an earlier release time than $J_{k,a}$, the condition holds. Therefore, according to Lemma 4, every earlier released job meets its deadline. Now we prove that the condition also holds for job $J_{k,a}$.

For job $J_{k,a}$, if there is no carry-in work from jobs released earlier than $J_{k,a}$, so that $R^{k,a} = 0$, the property trivially holds. Otherwise, there is at least one unfinished job (a job with carry-in work) at the release time of $J_{k,a}$.

We now define $J_{j,b}$ as the job with the earliest release time among all the unfinished jobs at the time that $J_{k,a}$ was released. For example, at release time $r_{3,1}$ of $J_{3,1}$ in Figure 2, both $J_{1,2}$ and $J_{2,1}$ are unfinished, but $J_{2,1}$ has the earliest release time. By the inductive assumption, the carry-in work $R^{j,b}$ at the release time of job $J_{j,b}$ is bounded by

$$R^{j,b} \leq \sum_i u_i \alpha_i^{j,b} + m \max_i (\alpha_i^{j,b}) \tag{7}$$

Let t be the number of time steps between the release time $r_{j,b}$ of $J_{j,b}$ and the release time $r_{k,a}$ of $J_{k,a}$.

$$t = r_{k,a} - r_{j,b}$$

Note that $J_{j,b}$ has not finished at time $r_{k,a}$, but by assumption it can meet its deadline. Therefore its absolute deadline $d_{j,b}$ is later than the release time $r_{k,a}$. So, by eq.(5)

$$t + \alpha_j^{k,a} = r_{k,a} - r_{j,b} + \alpha_j^{k,a} = d_{j,b} - r_{j,b} = D_j \tag{8}$$

In Figure 2, $t + \alpha_1^{3,1} = r_{3,1} - r_{2,1} + \alpha_1^{3,1} = d_{2,1} - r_{2,1} = D_2$.

For each task τ_i, let n_i^t be the number of jobs that are released after the release time $r_{j,b}$ of $J_{j,b}$ before the release time $r_{k,a}$ of $J_{k,a}$. The last such job may have a deadline after the release time of $r_{k,a}$, but its relase time is before $r_{k,a}$. In other words, n_i^t is the number of jobs of task τ_i, which fall entirely into the time interval $[r_{j,b}, r_{k,a} + D_i]$. By definition of $\alpha_i^{k,a}$, to job $J_{k,a}$, the deadline of the unfinished job of task i is $r_{k,a} + \alpha_i^{k,a}$. Therefore, for every τ_i,

$$\alpha_i^{j,b} + n_i^t D_i \leq r_{k,a} + \alpha_i^{k,a} - r_{j,b} = t + \alpha_i^{k,a} \tag{9}$$

As in the example in Figure 2, one entire job of task 1 falls within $[r_{2,1}, r_{3,1} + D_1]$, making $n_1^t = 1$ and $d_{1,2} = r_{3,1} + \alpha_1^{3,1}$. Also, since $d_{1,1} \leq r_{1,2}$, $\alpha_1^{2,1} + n_1^t D_1 = \alpha_1^{2,1} + D_1 \leq d_{1,2} - r_{2,1} = r_{3,1} + \alpha_1^{3,1} - r_{2,1} = t + \alpha_1^{3,1} \leq t + D_1$.

Compare between t and $\alpha_j^{k,a}$. When $t \leq \frac{1}{2} D_j$, by eq.(8), $\alpha_j^{k,a} = D_j - t \geq \frac{1}{2} D_j \geq t$. There are two cases:

Case 1: $t \leq \frac{1}{2} D_j$ and hence $\alpha_j^{k,a} \geq t$:

Since by definition $J_{j,b}$ is the earliest carry-in job, other carry-in jobs to $J_{k,a}$ are released after the release time of $J_{j,b}$ and therefore are not carry-in jobs to $J_{j,b}$. In other words, the carry-in jobs to $J_{j,b}$ must have been finished before the release time of $J_{k,a}$, which means that the carry-in work $R^{j,b}$ is not part of the carry-in work $R^{k,a}$. So the carry-in work $R^{k,a}$ is the sum of those released later than $J_{j,b}$

$$R^{k,a} = \sum_i u_i n_i^t D_i \leq \sum_i u_i (t + \alpha_i^{k,a}) \qquad \text{(from eq.(9))}$$

By assumption of case 1, $t \leq \alpha_j^{k,a} \leq \max_i \left(\alpha_i^{k,a} \right)$. Hence, replace $\sum_i u_i$ with m using eq.(2), we can prove that

$$R^{k,a} \leq \sum_i u_i \alpha_i^{k,a} + m \max_i \left(\alpha_i^{k,a} \right)$$

Case 2: $t > \frac{1}{2} D_j$:

Since $J_{j,b}$ has not finished executing at the release time of $J_{k,a}$, the total number of incomplete sub-steps during the t time steps $(r_{j,b}, r_{k,a}]$ is less than L_j. Therefore, the total work done during this time is at least F^t where

$$\begin{aligned} F^t &= bmt - (m-1)L_j \qquad \text{(from Lemma 2)} \\ &\geq bmt - (m-1)D_j \qquad \text{(from eq.(1))} \end{aligned}$$

The total amount of work from jobs that are released in time interval $(r_{j,b}, r_{k,a}]$ (i.e, entire jobs that fall in between the release time of job $J_{j,b}$ and the release time of job $J_{k,a}$ plus its deadline) is $\sum_i u_i n_i^t D_i$, by the definition of n_i^t. The carry-in work $R^{k,a}$ at the release time of job $J_{k,a}$ is the sum of the carry-in work $R^{j,b}$ and newly released work $\sum_i u_i n_i^t D_i$ minus the finished work during time interval t, which is

$$\begin{aligned} R^{k,a} &= R^{j,b} + \sum_i u_i n_i^t D_i - F^t \\ &\leq R^{j,b} + \sum_i u_i n_i^t D_i - (bmt - (m-1)D_j) \end{aligned}$$

By our assumption in eq.(7), we can replace $R^{j,b}$, reorganize the formula and get

$$\begin{aligned} R^{k,a} &\leq \sum_i u_i \alpha_i^{j,b} + m \max_i \left(\alpha_i^{j,b} \right) \\ &\quad + \sum u_i n_i^t D_i - bmt + (m-1)D_j \\ &\leq \sum_i u_i \left(\alpha_i^{j,b} + n_i^t D_i \right) + m \max_i \left(\alpha_i^{j,b} \right) \\ &\quad - bmt + (m-1)D_j \end{aligned}$$

According to eq.(9), we can replace $\alpha_i^{j,b} + n_i^t D_i$ with $t + \alpha_i^{k,a}$, reorganize the formula and get

$$\begin{aligned} R^{k,a} &\leq \sum_i u_i \left(t + \alpha_i^{k,a}\right) + m \max_i \left(\alpha_i^{j,b}\right) \\ &\quad -bmt + (m-1)D_j \\ &\leq \left(\sum_i u_i \left(t + \alpha_i^{k,a}\right) - mt\right) \\ &\quad +m \max_i \left(\alpha_i^{j,b}\right) + (m-1)D_j - (b-1)mt \end{aligned}$$

Using eq.(2) to replace m with $\sum_i u_i$ in the first item, using eq.(6) to get $\max_i \left(\alpha_i^{j,b}\right) \leq D_j$ and to replace $\max_i(\alpha_i^{j,b})$ with D_j in the second item, and since $t > \frac{1}{2}D_j$,

$$\begin{aligned} & R^{k,a} \\ \leq & \sum_i u_i \alpha_i^{k,a} + mD_j + (m-1)D_j - (b-2)mt - mt \\ \leq & \sum_i u_i \alpha_i^{k,a} + mD_j - mt + 2(m-1)t - (b-2)mt \\ \leq & \sum_i u_i \alpha_i^{k,a} + m(D_j - t) + 0 \qquad (\text{since } b \geq 4 - \tfrac{2}{m}) \\ \leq & \sum_i u_i \alpha_i^{k,a} + m\alpha_j^{k,a} \qquad (\text{from eq.(8)}) \end{aligned}$$

Finally, since $\alpha_j^{k,a} \leq \max_i \left(\alpha_i^{k,a}\right)$, we can prove that

$$R^{k,a} \leq \sum_i u_i \alpha_i^{k,a} + m \max_i \left(\alpha_i^{k,a}\right)$$

Therefore, by induction, if the processor speed $b \geq 4 - \frac{2}{m}$, for every $J_{k,a}$ in task set

$$R^{k,a} \leq \sum_i u_i \alpha_i^{k,a} + m \max_i \left(\alpha_i^{k,a}\right) \quad \blacksquare$$

From Lemmas 4 and 5, we can easily derive the following capacity augmentation bound theorem.

Theorem 1. *If, on unit speed processors, the utilization of a sporadic task set is at most m, and the critical path length of each job is at most its deadline, then the task set can meet all their implicit deadlines on m processors of speed $4 - \frac{2}{m}$.*

Theorem 1 proves the speed-up factor of GEDF and it can also be restated as follows:

Corollary 6. *If a sporadic task set τ with implicit deadlines satisfies the condition that its total utilization is no more than $1/(4 - \frac{2}{m})$ of the total system utilization m and for every task $\tau_i \in \tau$, the critical path length L_i is smaller than $1/(4 - \frac{2}{m})$ of its period D_i, then GEDF can successfully schedule task set τ.*

V. Resource Augmentation Bound of $2 - \frac{1}{m}$

In this section, we prove the resource augmentation bound of $2 - \frac{1}{m}$ for GEDF scheduling of arbitrary deadline tasks.

For sake of discussion, we convert the DAG representing of task into an equivalent DAG where each sub-node does $\frac{1}{m}$ unit of work. An example of this transformation of task 1 in Figure 1 is shown in job W_1 in Figure 3 (see the upper job). A node with work w is split into a chain of mw sub-nodes with work $\frac{1}{m}$. For example, since in Figure 3 $m = 2$, node W_1^1 with worst case execution time of 1 is split into 2 sub-nodes $W_1^{1,1}$ and $W_1^{1,2}$ with length $\frac{1}{2}$. The original incoming edges come into the first node of the chain, while the outgoing edges leave the last node of the chain. This transformation does not change any other characteristic of the DAG, and the scheduling does not depend on this step — the transformation is done only for clarity of the proof.

First, some definitions. Since the GEDF scheduler runs on processors of speed $2 - \frac{1}{m}$, each step under GEDF can be divided into $(2m-1)$ sub-steps of length $\frac{1}{m}$. In each sub-step, each processor can do $\frac{1}{m}$ unit of work (i.e. execute one sub-node). In a GEDF scheduler, on an incomplete step, all ready nodes are executed (Observation 7). As in Section IV, we say that a sub-step is ***complete*** if all processors are busy, and ***incomplete*** otherwise. For each sub-step t, we define $\mathcal{F}_{\mathcal{I}}(t)$ as the set of sub-nodes that have *finished* executing under an ideal scheduler after sub-step t, $\mathcal{R}_{\mathcal{I}}(t)$ as the set of sub-nodes that are *ready* (all their predecessors have been executed) to be executed by the ideal scheduler before sub-step t, and $\mathcal{D}_{\mathcal{I}}(t)$ as the set of sub-nodes completed by the ideal scheduler in sub-step t. Note that $\mathcal{D}_{\mathcal{I}}(t) = \mathcal{R}_{\mathcal{I}}(t) \cap \mathcal{F}_{\mathcal{I}}(t)$. We similarly define $\mathcal{F}_{\mathcal{G}}(t)$, $\mathcal{R}_{\mathcal{G}}(t)$, and $\mathcal{D}_{\mathcal{G}}(t)$ for GEDF scheduler.

Observation 7. *The GEDF scheduler completes all the ready nodes in an incomplete sub-step. That is,*

$$\mathcal{D}_{\mathcal{G}}(t) = \mathcal{R}_{\mathcal{G}}(t), \text{ if } t \text{ is incomplete sub-step,} \tag{10}$$

Note for the ideal scheduler, each original step consists of m sub-steps, while for GEDF with speed $2 - \frac{1}{m}$ each step consists of $2m-1$ sub-steps. For example, in Figure 3 for step t_1, there are two sub-steps $t_{1(1)}$ and $t_{1(2)}$ under ideal scheduler, while under GEDF there is an additional $t_{1(3)}$ (since $2m - 1 = 3$).

Theorem 2. *If an ideal scheduler can schedule a task set τ (periodic or sporadic tasks with arbitrary deadlines) on a unit-speed system with m identical processors, then global EDF can schedule τ on m processors of speed $2 - \frac{1}{m}$.*

Proof: In a GEDF scheduler, on an incomplete sub-step, all ready sub-nodes are executed (Observation 7). Therefore, after an incomplete sub-step, GEDF must have finished all the released sub-nodes and hence must have done at least as much work as the ideal scheduler. Thus, for brevity of our proof, we leave out any time interval when all processors under GEDF are idling, since at this time GEDF has finished all available work and at this time the Theorem is obviously true. We define time 0 as the first instant when not all processors are idling under GEDF and time t as any time such that for every sub-steps during time interval $[0, t]$ at least one processor under GEDF is working. Therefore for every incomplete sub-step GEDF will finish at least 1 sub-node (i.e. $\frac{1}{m}$ unit of work). We also define sub-step 0 as the last sub-step before time 0 and hence by definition,

$$\mathcal{F}_{\mathcal{G}}(0) \supseteq \mathcal{F}_{\mathcal{I}}(0) \text{ and } |\mathcal{F}_{\mathcal{G}}(0)| \geq |\mathcal{F}_{\mathcal{I}}(0)| \tag{11}$$

For each time $t \geq 0$, we now prove the following: If the ideal unit-speed system can successfully schedule all tasks with deadlines in the time interval $[0, t]$, then on speed $2 - \frac{1}{m}$ processors, so can GEDF. Note again that during the interval $[0, t]$ an ideal scheduler and GEDF have tm and $2tm - t$ sub-steps respectively.

Case 1: In $[0, t]$, GEDF has at most tm incomplete sub-steps.

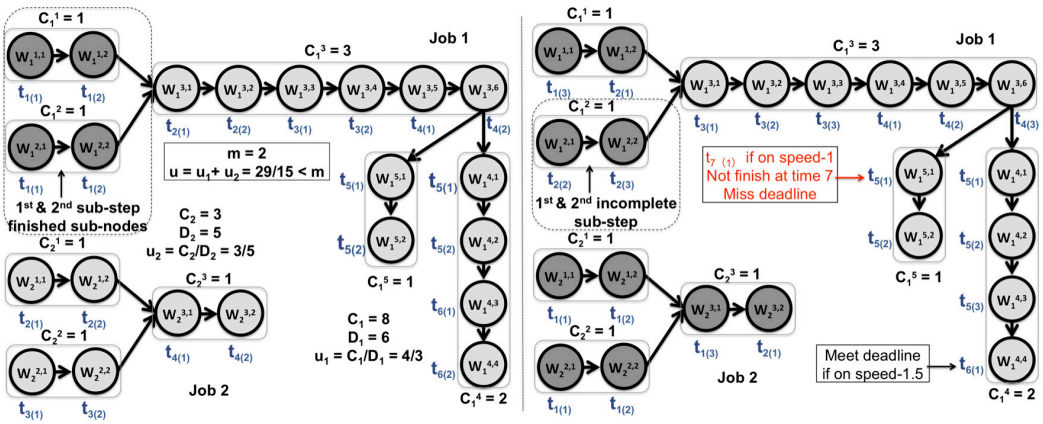

Fig. 3: Example of tasks execution: (left) under unit-speed ideal scheduler; (right) under 2-speed GEDF

Since there are at least $(2tm - t) - tm = tm - t$ complete steps, the system can complete $|\mathcal{F}_{\mathcal{G}}(t)| - |\mathcal{F}_{\mathcal{G}}(0)| \geq m(tm - t) + (tm) = tm^2$ work, since each complete sub-step can finish executing m sub-nodes and each incomplete sub-step can finish executing at least 1 sub-node. We define $I(t)$ as the set of all sub-nodes from jobs with absolute deadlines no later than t. Since the ideal scheduler can schedule this task set, we know that $|I(t)| - |\mathcal{F}_{\mathcal{I}}(0)| \leq mt * m = tm^2$, since the ideal scheduler can only finish at most m sub-nodes in each sub-step and during $[0, t]$ there are mt sub-steps for the ideal scheduler. Hence, we have $|\mathcal{F}_{\mathcal{G}}(t)| - |\mathcal{F}_{\mathcal{G}}(0)| \geq |I(t)| - |\mathcal{F}_{\mathcal{I}}(0)|$. By eq.(11), we get $|\mathcal{F}_{\mathcal{G}}(t)| \geq |I(t)|$. Note that jobs in $I(t)$ have earlier deadlines than the other jobs, so under GEDF, no other jobs can interfere with them. The GEDF scheduler will never execute other sub-nodes unless there are no ready sub-nodes from $I(t)$. Since $|\mathcal{F}_{\mathcal{G}}(t)| \geq |I(t)|$, i.e. the number of sub-nodes finished by GEDF is at least the amount in $I(t)$, this implies that GEDF must have finished all sub-nodes in $I(t)$. Therefore, GEDF can meet all deadlines since it has finished all work that needed to be done by time t.

Case 2: In $[0, t]$, the GEDF system has more than tm incomplete sub-steps.

For each integer s we define $f(s)$ as the first time instant such that the number of incomplete sub-steps in interval $[0, f(s)]$ is exactly s. Note that the sub-step $f(s)$ is always incomplete, since otherwise it wouldn't be the first such instant. We show, via induction, that $\mathcal{F}_{\mathcal{I}}(s) \subseteq \mathcal{F}_{\mathcal{G}}(f(s))$. In other words, after $f(s)$ sub-steps, GEDF has completed all the nodes that the ideal scheduler has completed after s sub-steps.

Base Case: For $s = 0$, $f(s) = 0$. By eq.(11), the claim is vacuously true.

Inductive Step: Suppose that for $s-1$ the claim $\mathcal{F}_{\mathcal{I}}(s-1) \subseteq \mathcal{F}_{\mathcal{G}}(f(s-1))$ is true. Now, we prove that $\mathcal{F}_{\mathcal{I}}(s) \subseteq \mathcal{F}_{\mathcal{G}}(f(s))$.

In $(s-1, s]$, the ideal system has exactly 1 sub-step. So,

$$\mathcal{F}_{\mathcal{I}}(s) = \mathcal{F}_{\mathcal{I}}(s-1) \cup \mathcal{D}_{\mathcal{I}}(s) \subseteq \mathcal{F}_{\mathcal{I}}(s-1) \cup \mathcal{R}_{\mathcal{I}}(s) \quad (12)$$

Since $\mathcal{F}_{\mathcal{I}}(s-1) \subseteq \mathcal{F}_{\mathcal{G}}(f(s-1))$, all the sub-nodes that are ready before sub-step s for the ideal scheduler, will either have already been executed or are also ready for the GEDF scheduler one sub-step after sub-step $f(s-1)$; that is,

$$\mathcal{F}_{\mathcal{I}}(s-1) \cup \mathcal{R}_{\mathcal{I}}(s) \subseteq \mathcal{F}_{\mathcal{G}}(f(s-1)) \cup \mathcal{R}_{\mathcal{G}}(f(s-1)+1) \quad (13)$$

For GEDF, from sub-step $f(s-1)+1$ to $f(s)$, all the ready sub-nodes with earliest deadlines will be executed and then new sub-nodes will be released into the ready set. Hence,

$$\begin{aligned} & \mathcal{F}_{\mathcal{G}}(f(s-1)) \cup \mathcal{R}_{\mathcal{G}}(f(s-1)+1) \\ \subseteq\ & \mathcal{F}_{\mathcal{G}}(f(s-1)+1) \cup \mathcal{R}_{\mathcal{G}}(f(s-1)+2) \\ \subseteq\ & \ldots \subseteq \mathcal{F}_{\mathcal{G}}(f(s)-1) \cup \mathcal{R}_{\mathcal{G}}(f(s)) \end{aligned} \quad (14)$$

Since sub-step $f(s)$ for GEDF is always incomplete,

$$\begin{aligned} & \mathcal{F}_{\mathcal{G}}(f(s)) \\ =\ & \mathcal{F}_{\mathcal{G}}(f(s)-1) \cup \mathcal{D}_{\mathcal{G}}(f(s)) \\ =\ & \mathcal{F}_{\mathcal{G}}(f(s)-1) \cup \mathcal{R}_{\mathcal{G}}(f(s)) \quad \text{(from eq.(10))} \\ \supseteq\ & \mathcal{F}_{\mathcal{G}}(f(s-1)) \cup \mathcal{R}_{\mathcal{G}}(f(s-1)+1) \quad \text{(from eq.(14))} \\ \supseteq\ & \mathcal{F}_{\mathcal{I}}(s-1) \cup \mathcal{R}_{\mathcal{I}}(s) \quad \text{(from eq.(13))} \\ \supseteq\ & \mathcal{F}_{\mathcal{I}}(s) \quad \text{(from eq.(12))} \end{aligned}$$

By time t, there are mt sub-steps for the ideal scheduler, so GEDF must have finished all the nodes executed by the ideal scheduler at sub-step $f(mt)$. Since there are exactly mt incomplete sub-steps in $[0, f(mt)]$ and since the number of incomplete sub-steps by time t is at least mt, the time $f(mt)$ is no later than time t. Since the ideal system does not miss any deadline by time t, GEDF also meets all deadlines. ■

An Example Providing Intuition of the Proof

We provide an example in Figure 3 to illustrate the proof of Case 2 and compare the execution trace of an ideal scheduler (this scheduler is only considered "ideal" in the sense that it makes all the deadlines) and GEDF. In addition to task 1 from Figure 1, task 2 consists of two nodes connected to another node, all with execution time of 1 (each split into 2 sub-nodes in the figure). All tasks are released by time t_0. The system has 2 cores, so GEDF has resource augmentation bound of 1.5. On the left side is the execution trace for the ideal scheduler on unit-speed cores, while the right side shows the execution trace under GEDF on speed 2 cores. One step is divided into 2 and 3 sub-steps, representing the speedup of 1 and 1.5 for the ideal scheduler and GEDF respectively.

Since the critical path length of task 1 is equal to its deadline, intuitively it should be executed immediately even though it has the latest deadline. That is exactly what the ideal scheduler does. However, GEDF (which does not take critical path length into consideration) will prioritize task 2

first. If GEDF is only on a unit-speed system, task 1 will miss deadline. However, when GEDF gets speed-1.5 processors, all jobs are finished in time. To illustrate Case 2 of the above theorem, consider $s = 2$. Since $t_{2(3)}$ is the second incomplete sub-step by GEDF, $f(s) = 2(3)$. All the nodes finished by the ideal scheduler after second sub-step (shown at left in dark grey) have also been finished under GEDF by step $t_{2(3)}$ (shown at right in dark grey).

Capacity Augmentation Bound is greater than $(3+\sqrt{5})/2$

While the above proof guarantees a bound, since the ideal scheduler is not known, given a task set, we cannot tell if it is feasible on speed-1 processors. Therefore, we cannot tell if it is schedulable by GEDF on processors with speed $2 - \frac{1}{m}$.

One standard way to prove resource augmentation bounds is to use lower bounds on the ideal scheduler, such as Inequalities 1 and 2. As previously stated, we call the resource augmentation bound proven using these lower bounds a ***capacity augmentation bound*** in order to distinguish it from the augmentation bound described above. To prove a capacity augmentation bound of b under GEDF, one must prove that if Inequalities 1 and 2 hold for a task set on m unit-speed processors, then GEDF can schedule that task set on m processors of speed b. Hence, the capacity augmentation bound is also an easy schedulability test.

First, we demonstrate a counter-example to show proving a capacity augmentation bound of 2 for GEDF is impossible.

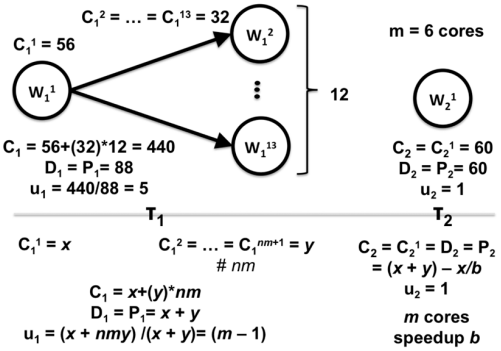

Fig. 4: Structure of the task set that demonstrates GEDF does not provide a capacity augmentation bound less than $(3+\sqrt{5})/2$

In particular, we provide a task set that satisfies inequalities 1 and 2, but cannot be scheduled on m processors of speed 2 by GEDF in Figure 4. In this example, as shown in Figure 5, $m = 6$. The task set has two tasks. All values are measured on a unit-speed system, shown in Figure 4. Task 1 has 13 nodes with total execution time of 440 and period of 88, so its utilization is 5. Task 2 is a single node, with execution time and implicit deadline both 60 and hence utilization of 1. Note the total utilization (6) is exactly equal to m, satisfying inequality 2. The critical path length of each task is equal to its deadline, satisfying inequality 1.

The execution trace of the task set on a 2-speed 6-core processor under GEDF is shown in Figure 5. The first task is released at time 0 and is immediately executed by GEDF. Since the system under GEDF is at speed 2, $W_1^{1,1}$ finishes

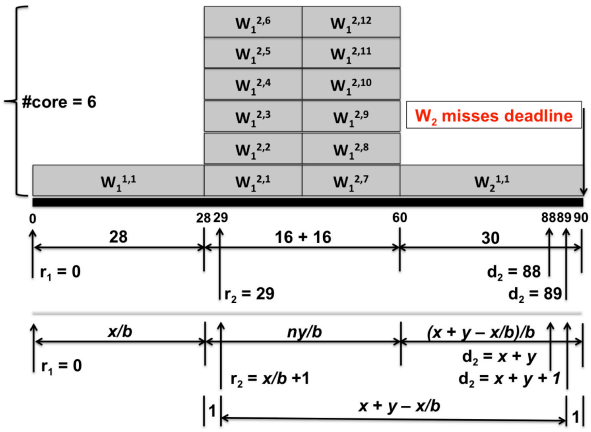

Fig. 5: Execution of the task set under GEDF at speed 2

executing at time 28. GEDF then executes 6 out of the 12 parallel nodes from task 1. At time 29, task 2 is released. However, its deadline is $r_2 + D_2 = 29 + 60 = 89$, later than deadline 88 of task 1. Nodes from task 1 are not preempted by task 2 and continue to execute until all of them finish their work at time 60. Task 1 successfully meets its deadline. The GEDF scheduler finally gets to execute task 2 and finishes it at time 90, so task 2 just fails to meet its deadline of 89. Note that this is not a counter-example for the resource augmentation bound shown in Theorem 2, since no scheduler can schedule this task set on unit-speed system either.

Second, we demonstrate that one can construct task sets that require capacity augmentation of at least $\frac{3+\sqrt{5}}{2}$ to be schedulable by GEDF. We generate task sets with two tasks whose structure depends on m, speedup factor b and a parallelism factor n, and show that for large enough m and n, the capacity augmentation required is at least $b \geq \frac{3+\sqrt{5}}{2}$. As in the lower part of Figure 4, task 1 is structured as a single node with work x followed by nm nodes with work y. Its critical path length is $x + y$ and so is its deadline. The utilization of task 1 is set to be $m - 1$, hence

$$m - 1 = \frac{x + nmy}{x + y} \tag{15}$$

Task 2 is structured as a single node with work and deadline equal to $x + y - \frac{x}{b}$ (hence utilization 1). Therefore, the total task utlization is m and Inequalities 1 and 2 are met. As the lower part of Figure 5 shows, Task 2 is released at time $\frac{x}{b} + 1$.

We want to generate a counter example, so want task 2 to barely miss the deadline by 1 sub-step. In order for this to occur, we must have

$$(x + y - \frac{x}{b}) + 2 = \frac{ny}{b} + \frac{1}{b}(x + y - \frac{x}{b}). \tag{16}$$

Reorganizing and combining eq.(15) and eq.(16), we get

$$(m-2)b^2 = ((3bn-b-n-b^2n+1)m+(b^2-2bn-1))y \tag{17}$$

In the above equation, for large enough m and n, we have $(3bn - b - n - b^2n + 1) > 0$, or

$$1 < b < \frac{3}{2} - \frac{1}{2n} + \frac{1}{2}\sqrt{5 - \frac{2}{n} + \frac{1}{n^2}} \tag{18}$$

So, there exists a counter-example for any speedup b which satisfies the above conditions. Therefore, the capacity augmentation required by GEDF is at least $\frac{3+\sqrt{5}}{2}$. The example above

with speedup of 2 comes from such a construction. Another example with speedup 2.5 can be obtained when $x = 36050$, $y = 5900$, $m = 120$ and $n = 7$.

VI. Evaluation

In this section, we describe our experimental evaluation the performance of GEDF and the robustness of our capacity augmentation bound.[1] We randomly generate task sets that fully load machines, and then simulate their execution on machines of increasing speed. The capacity augmentation bound for GEDF predicts that all task sets should be schedulable by the time the processor speed is increased to $4 - \frac{2}{m}$. In our simulation experiments, all task sets became schedulable before the speed reached 2.

We also compared GEDF with the only other known method that provides capacity bounds for scheduling multiple DAGs (with a DAG's utilization potentially more than 1) on multi-cores [4]. In this method, which we call ***DECOMP***, tasks are decomposed into sequential subtasks and then scheduled using GEDF.[2] We find that GEDF without decomposition performs better than DECOMP for most task sets.

A. Task Sets and Experimental Setup

We generate two types of DAG tasks for evaluation. For each method, we first fix the number of nodes n in the DAG and then add edges.

(1) Erdos-Renyi method $G(n, p)$ [36]: Each valid edge is added with probability p, where p is a parameter. Note that this method does not necessarily generate a connected DAG. Although the bound also does not require the DAG of a task to be fully connected, connecting them can make it harder to schedule. Hence, we modified it slightly in the last step, to add the fewest edges needed to make the DAG connected.

(2) Special synchronous task $L(n, m)$: As shown in Figure 4, task 1, synchronous tasks, in which highly parallel segments follow sequential segments make scheduling difficult for GEDF since they can cause deadline misses for other tasks. Therefore, we generate task sets with alternating sequential and highly parallel segments. Tasks in $L(n, m)$ (m is the number of processors) are generated in the following way. While the total number of nodes in the DAG is smaller than n, we add another sequential segment by adding a node, then generate the next parallel layer randomly. For each parallel layer, we uniformly generate a number t between 1 and $\lfloor \frac{n}{m} \rfloor$, and set the number of nodes in the segment to be $t * m$.

Given a task structure generated by either of the above methods, worst case execution times for individual nodes in the DAG are picked randomly between $[50, 500]$ The critical path length L_i for each task is then calculated. We then assign a period (equal to its deadline) to each task. Note that a valid deadline is at least the critical path length. Two types of periods were assigned to tasks

[1]Note that, due to the lack of a schedulability test, it is difficult to test the resource augmentation bound of $2 - \frac{1}{m}$ experimentally.

[2]For DECOMP, end-to-end deadline (instead of decomposed subtask's deadline) miss ratios were reported.

(1) Harmonic Period: In this method, all tasks have periods that are integral powers of 2. We first compute the smallest value a such that 2^a is larger than a task's critical path length L_i. We then randomly assign the period either 2^a, 2^{a+1} or 2^{a+2} to generate tasks with varying utilization. All tasks are then released at the same time and simulated for the hyper-period of the tasks.

(2) Arbitrary Period: An arbitrary period is assigned in the form $(L_i + \frac{C_i}{0.5m}) * (1 + 0.25 * gamma(2, 1))$, where $gamma(2, 1)$ is the Gamma distribution [37] with $k = 2$ and $\theta = 1$. The formula is designed such that, for small m, tasks tend to have smaller utilization. This allows us to have a reasonable number of tasks in a task set for any value of m. Task sets are created by adding tasks to them until the total utilization reaches 99% of m. These task sets were simulated for 20 times the longest period in a task set.

Several parameters were varied to test the system: $G(n, p)$ vs $L(n, m)$ DAGs, different p for $G(n, p)$, Harmonic vs Arbitrary Periods, Numbers of Cores m (4, 8, 16, 32, 64). For each setting, we generated a 1000 task sets. We first simulated the task sets for each setting on processors of speed 1, and increased the speed in steps of 0.2. For each setting, we measured the ***failure rate*** — the number of task sets where *any* task missed its deadline.

B. Experiment Results

1) Erdos-Renyi Method: For this method, we generate two types of task sets: (1) **Fixed p task sets**: In this setting, all task sets have the same p. We varied the values of p from 0.01 to 0.9. (2) **Random p task sets**: We also generated task sets where each task has a different, randomly picked, value of p.

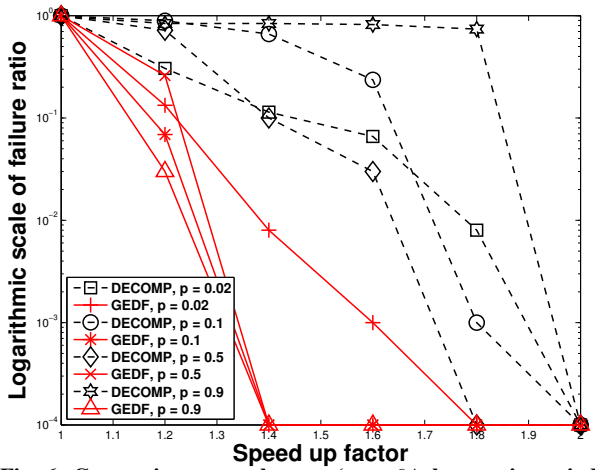

Fig. 6: Comparison as p changes ($m = 64$, harmonic period)

Figure 6 shows the failure rate for fixed-p task sets as we varied p and kept m constant at 64. GEDF without decomposition outperforms DECOMP for all settings of p. It appears that GEDF has the hardest time when $p \leq 0.1$, where tasks are more sequential. But even then, all tasks are schedulable with speed 1.8. At $p > 0.1$, GEDF never requires speed more than 1.4, while DECOMP often requires a speed of 2 to schedule all task sets. Trends are similar for other values of m.

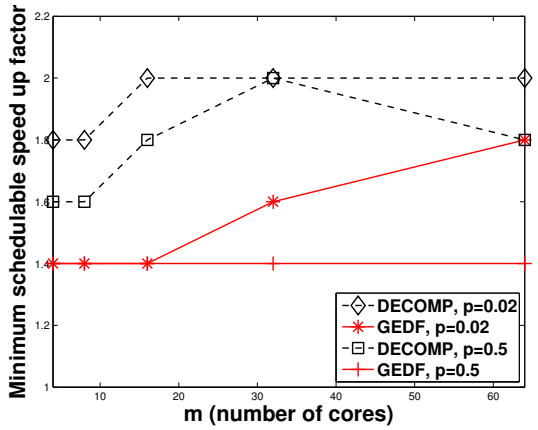

Fig. 7: Minimum schedulable speed up as m changes (harmonic period)

In addition to varying p, we also varied the number of cores m. In Figure 7, among all different combinations we show the minimum schedulable speed up as m increases under $p = 0.02$ and $p = 0.5$. Results for other p and m are similar. This shows that GEDF without decomposition generally needs smaller speed up to schedule the same task sets. The increase of m in this experiment setting makes task sets harder to schedule for many cases, but larger m is not always worse.

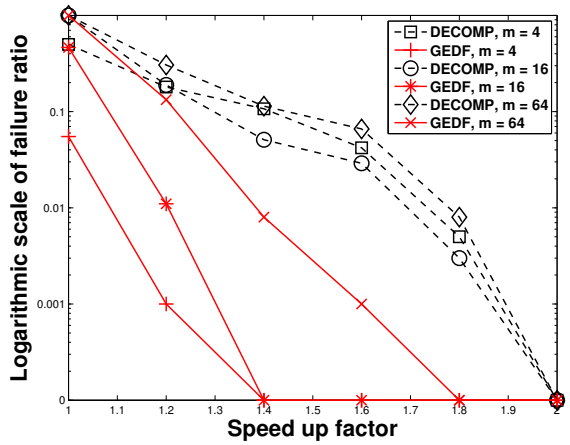

Fig. 8: Comparison as m changes ($p = 0.02$, harmonic period)

For the rest of the figures, we only show results with $m = 4$, 16 and 64, since the trends for $m = 8$ and 32 are similar.

Figure 8 shows the failure ratio of the fixed-p task sets as we kept p constant at 0.02 and varied m. Again, GEDF outperforms DECOMP for all settings. When $m = 4$, GEDF can schedule all task sets at speed 1.4. The increase of m does not influence DECOMP much, while it becomes slightly harder for GEDF to schedule a few (out of 1000) task sets.

In addition to harmonic periods, we also run experiments for arbitrary periods for all different types of task sets. As the comparison between Figure 9 and 8 suggest, the trends for arbitrary and harmonic periods are similar. Though it appears that tasks with arbitrary periods are easier to schedule using, especially for GEDF. This is at least partially explained by the observation that, with harmonic periods, many tasks have

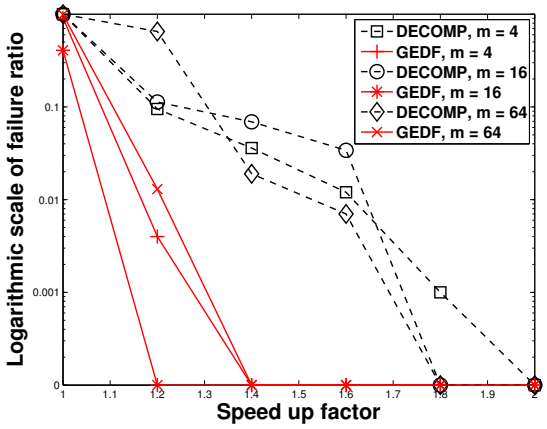

Fig. 9: Comparison as m changes ($p = 0.02$, arbitrary period)

the same deadline, making it difficult for GEDF to distinguish between them. This trends holds for other types of task sets. So for brevity, we only show the experimental results for harmonic periods for the other experiments.

Figure 10 shows the failure ratio for GEDF and DECOMP for task sets where p is not fixed, but is randomly generated for each task, as we vary m. Again, GEDF outperforms DECOMP. Note, however, that it appears that GEDF appears to have a harder time here than in the fixed p experiment.

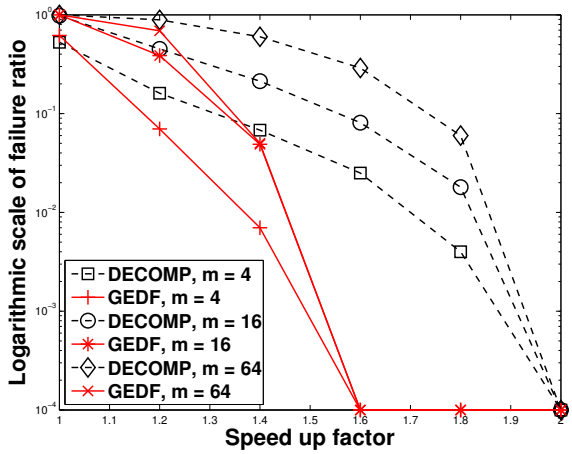

Fig. 10: Comparison as m changes(random p, harmonic period)

2) Synchronous Method: Figure 11 shows the comparison between GEDF and DECOMP with varying m for synchronous task sets. In this case, the failure ratio for GEDF is higher than for task sets generated with the Erdos-Renyi Method. We can also see that sometimes DECOMP outperforms GEDF in terms of failure ratio and required speed up. This indicates that synchronous tasks with highly parallel segments are indeed more difficult for GEDF to schedule. However, even in this case, we never require a speedup of more than 2. Even though Figure 4 demonstrates that there exist task sets that require speedup of more than 2, such pathological task sets never appeared in our randomly generated sample.

Results indicate that GEDF performs better than predicted by the capacity augmentation bound. For most task sets, GEDF is better than DECOMP. Only among task sets intentionally

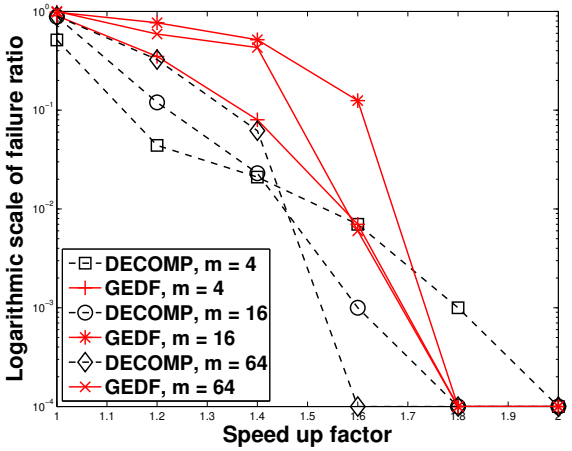

Fig. 11: Comparison as m changes ($L(n,m)$ tasks, harmonic period)

designed to be harder for GEDF to schedule, DECOMP performs slightly better.

VII. Conclusions

In this paper, we have presented the best bounds known for parallel tasks represented as DAGs under GEDF. In particular, we proved that GEDF provides a resource augmentation bound of $2 - 1/m$ for sporadic task sets with arbitrary deadlines and a capacity augmentation bound of $4 - 2/m$ with implicit deadlines. The capacity augmentation bound also serves as a simple schedulability test, namely, a task set is schedulable on m processors if (1) m is at least $4 - \frac{2}{m}$ times its total utilization, and (2) the implicit deadline of each task is at least $4 - \frac{2}{m}$ times its critical path length. We also present simulation results indicating this bound is safe; in fact, we never saw a required capacity augmentation of more than 2 on randomly generated task sets in our simulations. Possible directions of future work: First, we would like to extend the capacity augmentation bounds to constrained and arbitrary deadlines; In addition, while we prove that a capacity augmentation bound of more than $\frac{3+\sqrt{5}}{2}$ is needed, there is still a gap between this lower bound and the upper bound of $(4 - 2/m)$ for capacity augmentation, which we would like to close.

Acknowledgment

This research was supported in part by NSF grant CCF-1136073 (CPS)

References

[1] A. Maghareh, S. J. Dyke, A. Prakash, G. Bunting, and P. Lindsay, "Evaluating modeling choices in the implementation of real-time hybrid simulation," in *EMI/PMC 2012*.

[2] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *RTSS '10*.

[3] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *RTSS '11*.

[4] A. Saifullah, D. Ferry, K. Agrawal, C. Lu, and C. Gill, "Real-time scheduling of parallel tasks under a general dag model," Washington University in St Louis, USA, Tech. Rep. WUCSE-2012-14, 2012.

[5] D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu, "A real-time scheduling service for parallel tasks," in *RTSS '13*.

[6] J. Lelli, D. Faggioli, T. Cucinotta, and S. Superiore, "An efficient and scalable implementation of global edf in linux," in *OSPERT '11*.

[7] B. B. Brandenburg and J. H. Anderson, "On the implementation of global real-time schedulers," in *RTSS '09*.

[8] S. Baruah, V. Bonifaciy, A. Marchetti-Spaccamelaz, L. Stougiex, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *RTSS '12*.

[9] C. Liu and J. Anderson, "Supporting soft real-time parallel applications on multicore processors," in *RTCSA '12*.

[10] N. Fisher, J. Goossens, and S. Baruah, "Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible," *Real-Time Syst.*, vol. 45, no. 1-2, pp. 26–71, 2010.

[11] C. D. Polychronopoulos and D. J. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers," *IEEE Transactions on Computers*, vol. C-36, no. 12, pp. 1425–1439, 1987.

[12] M. Drozdowski, "Real-time scheduling of linear speedup parallel tasks," *Inf. Process. Lett.*, vol. 57, no. 1, pp. 35–40, 1996.

[13] X. Deng, N. Gu, T. Brecht, and K. Lu, "Preemptive scheduling of parallel jobs on multiprocessors," in *SODA '96*.

[14] K. Agrawal, C. E. Leiserson, Y. He, and W. J. Hsu, "Adaptive work-stealing with parallelism feedback," *ACM Trans. Comput. Syst.*, vol. 26, September 2008.

[15] J. M. Calandrino and J. H. Anderson, "On the design and implementation of a cache-aware multicore real-time scheduler," in *ECRTS '09*.

[16] J. H. Anderson and J. M. Calandrino, "Parallel real-time task scheduling on multicore platforms," in *RTSS '06*.

[17] Q. Wang and K. H. Cheng, "A heuristic of scheduling parallel tasks and its analysis," *SIAM J. Comput.*, vol. 21, no. 2, 1992.

[18] O.-H. Kwon and K.-Y. Chwa, "Scheduling parallel tasks with individual deadlines," *Theor. Comput. Sci.*, vol. 215, no. 1-2, pp. 209–223, 1999.

[19] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comp. Surv.*, vol. 43, pp. 35:1–44, 2011.

[20] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *RTSS '01*, dec. 2001, pp. 193–202.

[21] A. Srinivasan and S. Baruah, "Deadline-based scheduling of periodic task systems on multiprocessors," *Information Processing Letters*, vol. 84, no. 2, pp. 93 – 98, 2002.

[22] J. Goossens, S. Funk, and S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-Time Systems*, vol. 25, no. 2, pp. 187–205, 2003.

[23] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *Parallel and Distributed Systems*, vol. 20, no. 4, pp. 553 –566, april 2009.

[24] S. Baruah and T. Baker, "Schedulability analysis of global edf," *Real-Time Systems*, vol. 38, no. 3, pp. 223–235, 2008.

[25] T. Baker and S. Baruah, "Sustainable multiprocessor scheduling of sporadic task systems," in *ECRTS '09*, july 2009, pp. 141 –150.

[26] J. Lee and K. G. Shin, "Controlling preemption for better schedulability in multi-core systems," in *RTSS '12*, Dec. 2012.

[27] S. Baruah, "Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors," *Computers, IEEE Transactions on*, vol. 53, no. 6, pp. 781–784, june 2004.

[28] M. Bertogna and S. Baruah, "Tests for global edf schedulability analysis," *J. Syst. Archit.*, vol. 57, no. 5, pp. 487–497, 2011.

[29] W. Y. Lee and H. Lee, "Optimal scheduling for real-time parallel tasks," *IEICE Trans. Inf. Syst.*, vol. E89-D, no. 6, pp. 1962–1966, 2006.

[30] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Inf. Process. Lett.*, vol. 106, no. 5, pp. 180–187, 2008.

[31] G. Manimaran, C. S. R. Murthy, and K. Ramamritham, "A new approach for scheduling of parallelizable tasks in real-time multiprocessor systems," *Real-Time Syst.*, vol. 15, no. 1, pp. 39–60, 1998.

[32] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *RTSS '09*.

[33] N. Fisher, S. Baruah, and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *ECRTS '06*.

[34] L. Nogueira and L. M. Pinho, "Server-based scheduling of parallel real-time tasks," in *International Conference on Embedded Software*, 2012.

[35] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic dag task model," in *ECRTS '13*.

[36] D. Cordeiro, G. Mouni, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *SIMUTools '10*.

[37] "Gamma distribution," http://en.wikipedia.org/wiki/Gamma_distribution.