Global EDF scheduling of directed acyclic graphs on multiprocessor systems

Manar Qamhieh, Frédéric Fauberteau, Laurent George, Serge Midonnet

Global EDF Scheduling of Directed Acyclic Graphs on Multiprocessor Systems

Manar Qamhieh
Université Paris-Est
qamhieh@univ-mlv.fr

Frédéric Fauberteau
ECE Paris - LACSC
fauberte@ece.fr

Laurent George
Université Paris-Est
laurent.george@univ-mlv.fr

Serge Midonnet
Université Paris-Est
midonnet@univ-mlv.fr

ABSTRACT

In this paper, we study the problem of real-time scheduling of parallel tasks represented by a Directed Acyclic Graph (DAG) on multiprocessor architectures. We focus on Global Earliest Deadline First scheduling of sporadic DAG tasksets with constrained-deadlines on a system of homogeneous processors. Our contributions consist in analyzing DAG tasks by considering their internal structures and providing a tighter bound on the workload and interference analysis. This approach consists in assigning a local offset and deadline for each subtask in the DAG. We derive an improved sufficient schedulability test w.r.t. an existing test proposed in the state of the art. Then we discuss the sustainability of this test.

1. INTRODUCTION

Uniprocessor platforms have been widely used in computer systems and applications for a long time. However, making processors smaller and faster has become more challenging for manufacturers recently due to heating and power problems. As a result, manufacturers are moving toward building multicore and multiprocessor systems, such as the 72-core processor of the TILE-Gx family from Tilera, and the 192-core processor released by ClearSpeed in 2008.

Unfortunately, the development of software is not as fast as the hardware, and many of the industrial applications are still built for single processor platforms. This prevents them from achieving the maximum performance of the multiprocessor platform. *Parallelism* is a programming technique used to perform calculations on multiple processors simultaneously. There are many parallel programming APIs used in practice, such as OpenMP and POSIX Threads.

Real-time systems are defined as the systems in which the correctness of the application depends on temporal constraints in addition to the correctness of the results. These systems have been studied thoroughly for uniprocessor platforms and many scheduling algorithms and analysis have been provided over the years. However, the shift from uniprocessor to multiporcessor platforms is a challenge in real-time systems, because of the additional execution and resource synchronization constraints between the different processors, and the management of job migrations and preemptions. However, adapting scheduling algorithms designed for uniprocessors to be used on multiprocessors usually reduces their performance. For example, the optimal[1] Earliest Deadline First (EDF) scheduling algorithm on uniprocessor systems loses its optimality when applied to multiprocessor systems.

Most studies in real-time scheduling have focused on sequential independent tasks for uniprocessor and multiprocessor systems, and fewer studies were done on parallel tasks. A parallel real-time task is a task that is allowed to execute on multiple processors at the same time, it either consists of identical threads that execute in parallel and synchronize the result at a merge point (as in fork-join model), or a collection of different subtasks that execute in parallel according to time and precedence constraints. This model is called the Directed Acyclic Graph (DAG). The schedulability of different models of parallel real-time tasks are shown later in Section 2.

The contribution of this paper is related to Global-EDF (G-EDF) scheduling in which job migrations are allowed. They are summarized as follows:

- a new and more precise analysis of the DAG tasks in the context of scheduling on homogeneous multiprocessor systems, while considering the internal structure and the execution flow of DAGs,
- an improved schedulability test for the G-EDF scheduling of sporadic constrained-deadline DAG tasks,
- a study of the sustainability of G-EDF scheduling policy and the proposed scheduling test.

The remainder of this paper is organized as follows. In Section 2, we represent a state-of-the-art methods relative to real-time parallel task scheduling on multiprocessor systems especially for a DAG model. The considered model and the used terminology are described in Section 3. The new analysis of the DAG tasks is described in Section 4.

[1]If there is an algorithm that can schedule a given taskset, then it is schedulable by the optimal algorithm.

RTNS 2013 , October 16 - 18 2013, Sophia Antipolis, France

http://dx.doi.org/10.1145/2516821.2516836.

A new schedulability test for G-EDF scheduling is derived from this analysis in Section 5. The sustainability property of this test is studied in Section 6. In Section 7, we study the performance of our feasibility test by simulation and we discuss the obtained results. Finally, we conclude this study and show future work in Section 8.

2. RELATED WORK

Many hard real-time scheduling algorithms and schedulability analyses on homogeneous multiprocessor systems have been proposed in the literature [11]. However, they focus on the traditional sequential independent real-time task model. The problem of scheduling sporadic tasksets on multiprocessor systems is more complicated. Moreover it has been shown NP-hard in the strong sense in [3]. This work focuses on the G-EDF scheduling algorithm and provides a sufficient schedulability test in polynomial time. A more efficient but more complex sufficient test that runs in pseudo-polynomial time has also been proposed.

Regarding parallel tasks, there are different models and each has its own advantages and limitations. First there is the Fork-join model, in which a parallel task is an alternating sequence of parallel and sequential segments, a stretching algorithm to execute the parallel segments as sequential as possible was proposed in [14]. It provided a resource augmentation bound of $3,42$ when the Deadline-Monotonic scheduling algorithm is used. Another parallel task model which integrates work-limited job parallelism was proposed in [18]. The authors investigated global scheduling for sporadic implicit-deadline taskset and they proposed a theoretically optimal scheduling algorithm.

A more general model of parallel tasks, called the multi-threaded segment model, has been studied in the literature. The multiprocessor scheduling of periodic tasksets with implicit deadlines of this model has been addressed in [17]. This work proved a resource augmentation bound of 4 for a G-EDF scheduling and 5 for Partitioned DM scheduling. The analysis has been extended to the DAG model and the same results can be applied. Another scheduling approach based on the response time analysis for this task model has been provided in [16] for soft real-time multi-core systems.

Most of the parallel task models presented earlier are considered special cases of the DAG model, in which a task is represented as a graph of nodes and directed relations. This model has been studied in [9] for the uniprocessor case. The authors considered a hybrid task set of periodic independent tasks and dependent sporadic graph tasks that execute only once. A graph task in this model consists of a set of tasks with precedence constraints and each task has a release time and deadline. They proposed an algorithm based on a modification of task parameters in order to avoid the dependencies between the tasks. Their algorithm is based on the same concept as our algorithm presented in Section 4 with few differences because of the characteristics of the model.

A G-EDF scheduling analysis of a single arbitrary-deadline DAG task has been studied in [3], in which the authors proved a resource augmentation bound of 2.

A capacity augmentation bound of $4 - \frac{2}{m}$ and a resource augmentation bound of $2 - \frac{1}{m}$ have been proposed recently for G-EDF scheduling of periodic implicit-deadline DAG tasksets in [15]. While a resource augmentation bound enables to gauge the distance between a given scheduling algorithm and a hypothetical optimal scheduler, a capacity augmentation bound can be used directly as a sufficient schedulability test.

3. MODEL AND TERMINOLOGY

We consider a set of n sporadic parallel real-time tasks, scheduled on a system of m identical processors. The taskset is denoted by $\tau = \{\tau_1, \ldots, \tau_n\}$ and the processor set is denoted by $\pi = \{\pi_1, \ldots, \pi_m\}$. Each parallel task τ_i, where $1 \leq i \leq n$, is represented by a DAG and consists of a set of nodes and directed relations. The nodes represent the execution requirements of the task, while the directed relations show the execution flow. A real-time DAG task τ_i is characterized by $(n_i, \{1 \leq j \leq n_i | \tau_{i,j}\}, G_i, D_i, T_i)$, where n_i is the number of subtasks of τ_i, the second parameter is the set of subtasks, G_i is the set of directed relations between subtasks, D_i is the relative deadline of τ_i and T_i is the minimum time interval between two successive jobs of τ_i (sporadic tasks). In this work, we consider constrained-deadline DAGs, which means that the deadline of each DAG cannot exceed its period ($D_i \leq T_i$). Each subtask $\tau_{i,j}$, where $1 \leq j \leq n_i$, has a Worst-Case Execution Time (WCET) denoted $C_{i,j}$. Let C_i denote the worst-case total execution time of a DAG τ_i, which is the sum of the WCET of all of its subtasks, $C_i = \sum_{\forall \tau_{i,j} \in \tau_i} C_{i,j}$.

According to the DAG model, the execution flow of the subtasks is constrained by their directed relations. A directed relation from subtask $\tau_{i,j}$ to $\tau_{i,k}$ means that $\tau_{i,k}$ can start its execution only if $\tau_{i,j}$ completes its own. In this case, we call subtask $\tau_{i,j}$ a **parent** subtask of $\tau_{i,k}$ and $\tau_{i,k}$ the **child** of $\tau_{i,j}$. Each subtask in a certain DAG could have multiple parents and children. A **starting** subtask is a parentless subtask, while an **ending** subtask is a childless subtask. A DAG task could have multiple starting and ending subtasks. Figure 1 shows an example of DAG task τ_1 which consists of six subtasks where $\tau_{1,1}$ is the starting subtask and $\tau_{1,6}$ is the ending subtask. The directed arrows between the subtasks in the figure represent the precedence constraints between the subtasks.

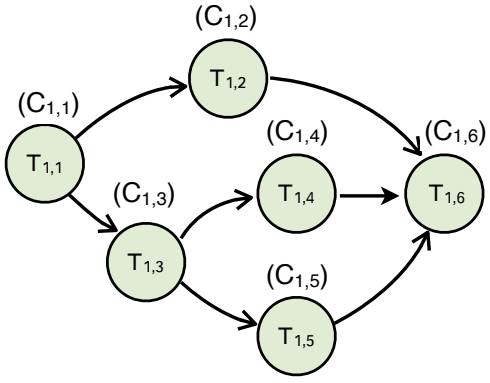


Figure 1: An example of DAG task consists of six subtasks.

The **critical path** of a DAG is defined as the longest path among all the other paths in the DAG if executed on a system of infinite number of processors. Let L_i be the length of the critical path of DAG τ_i. The utilization of a

DAG task is denoted by $u_i = \frac{C_i}{T_i}$. A DAG job is defined as $J_i = (r_i, d_i)$, where r_i is the release time of the job and d_i is its absolute deadline. A job of subtask $\tau_{i,j}$ is denoted by $J_{i,j}$ as well.

Regarding the feasibility of parallel tasks of the DAG model on a system of m processors, two necessary conditions can be defined as the following:

$$\sum_{\tau_i \in \tau} u_i \leq m$$

$$\forall \{\tau_i \in \tau\} : L_i \leq D_i$$

If any of these two conditions does not hold for a taskset of DAGs, then the taskset cannot be scheduled by any scheduling algorithm on m processors of speed 1. It is obvious that a system with m identical unit-speed processors cannot schedule a taskset of utilization higher than m. Also, the critical path length L_i of each DAG, which is the longest sequential path in DAG τ_i, should not exceed its deadline D_i.

4. DAG ANALYSIS

Parallel real-time tasks of the DAG model have particular characteristics due to precedence constraints between the subtasks and their activation dependencies. This is why scheduling DAG tasks on multiprocessor systems is challenging. As described in Section 3, a DAG task consists of a set of subtasks and directed relations, and the scheduling of DAG tasks depends on the dependencies of the subtasks and the structure of the DAGs.

A DAG job is said to be **ready** at time t if it is released at t, while a subtask job is said to be **ready** if its DAG job is ready and all the precedent subtasks have completed their execution. By default, the DAG model defines one individual temporal parameter for subtask $\tau_{i,j}$ which is the WCET $C_{i,j}$. The rest of the parameters — such as its deadline, period and offset — are inherited from its DAG τ_i and are shared with the other subtasks. This is why the schedulability of the DAGs is complicated. A DAG task τ_i can be treated as a single unit denoted by its total WCET C_i, its deadline D_i, its period T_i and its critical path length L_i, as in [3] and [15]. This assumption does not take into account the internal DAG structure nor the dependencies between the subtasks in the scheduling process. However, we show that ignoring the structure of the DAGs results in a pessimistic analysis.

Example. In Figure 2, we show an example which motivates the need for knowledge about the internal structure of DAG. Both DAG tasks τ_1 and τ_2 have the same external structure and number of subtasks, where $C_1 = C_2 = 6$, $D_1 = D_2 = 4$, $T_1 = T_2 = 4$ and $L_1 = L_2 = 4$. However, they differ in their execution flow. The first DAG τ_1 has subtask $\tau_{1,3}$ executed in parallel with subtask $\tau_{1,2}$, and they are both activated after $\tau_{1,1}$ completion. On the other hand, subtask $\tau_{2,3}$ of DAG τ_2 executes in parallel with $\tau_{2,1}$ and they are the starting subtasks of τ_2, while $\tau_{2,3}$ executes after they complete their activation.

We show in this example that it is not enough to build the scheduling decisions based on the external structure of the DAGs as it can lead to pessimistic scheduling decisions. Therefore, we propose in the next section an analysis of the DAG tasks to identify their subtasks to be included in the scheduling process.

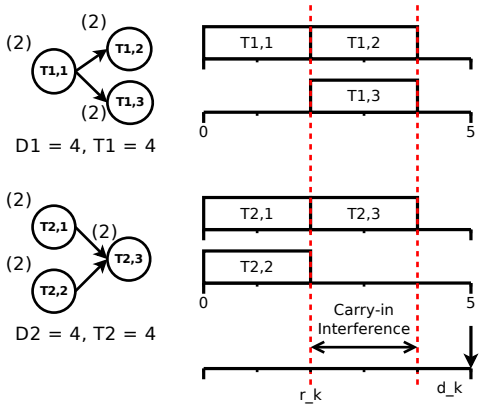


Figure 2: An example of two different DAG tasks with the same external structure.

4.1 Local offset and deadline for subtasks

In this section, we assign additional temporal parameters for the subtasks of the DAG tasks, so as to define their internal structure and improve their schedulability. Based on the structure of the DAG model, we provide the following definitions:

Definition 1. A **local offset** $O_{i,j}$ of subtask $\tau_{i,j}$ is defined as the earliest possible release time w.r.t the activation of DAG τ_i in which subtask $\tau_{i,j}$ can be ready, and this is the length of the longest path from the starting subtask in τ_i to $\tau_{i,j}$.

In order to define $O_{i,j}$, we take into consideration the precedence constraints of DAG τ_i. It is defined as the minimum duration that $\tau_{i,j}$ has to be activated while making sure that its predecessors are terminated. Following this definition, it is impossible for a subtask $\tau_{i,j}$ to be ready before $O_{i,j}$.

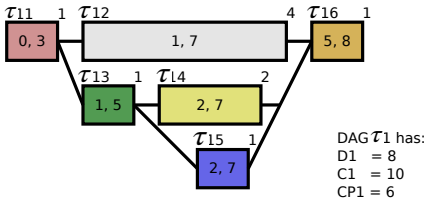
The calculation of the local offset of each subtask in the DAG is done assuming that the system has an infinite number of processors. In this scenario, all ready subtasks will execute as soon as possible and the response time of DAG will be equal to its critical path length. We apply a straightforward depth-first search algorithm shown in Algorithm 1 on each DAG task. This algorithm executes in linear time to calculate the offset of all of its subtasks.

Algorithm 1 Local offset algorithm

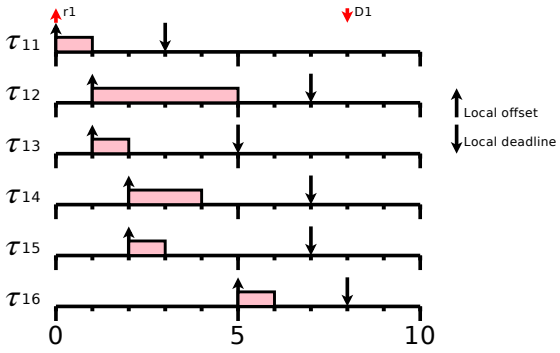
```
        ▷ Inputs: τ_i is a graph task, τ_{i,j} is a subtask in τ_i
procedure LOCAL_OFFSET(τ_{i,j})
    if τ_{i,j} = start_subtask(τ_i) then
        return r_i
    else
        return max_{τ_{i,k} ∈ Pred(τ_{i,j})} (
        LOCAL_OFFSET(τ_{i,k}) + C_{i,k})
    end if
end procedure
```

In Figure 3 we represent the DAG task τ_1, in which $D_1 = T_1 = 8$, and a total WCET $C_1 = 10$ from six subtasks, and we assume that τ_1 has no offset. In Figure 3(a) we show

the DAG task. Each subtask in the DAG is displayed with a square and its WCET is the number on the upper right corner. We calculate the local offset of each subtask $\tau_{1,j}$, where $1 \leq j \leq 6$. Subtask $\tau_{1,1}$ is the starting subtask in τ_1, ready at the release time of DAG τ_1. For its successor subtasks $\tau_{1,2}$ and $\tau_{1,3}$, their earliest offset is the WCET of their parent subtask $O_{1,2} = O_{1,3} = C_{1,1} = 1$, and it is impossible for these subtasks to be released earlier. Finally, the local release time $O_{1,6}$ of the ending subtask $\tau_{1,6}$ is the longest path from $\tau_{1,1}$ to $\tau_{1,6}$ which is $\{\tau_{1,1}, \tau_{1,2}\}$ and $O_{1,6} = 5$.



(a) A DAG task τ_1 consists of six subtasks and their directed relations



(b) The time diagram of each subtask in τ_1 shows their local release and deadline

Figure 3: An example of the analysis of local offsets and deadlines for a DAG.

Secondly, we define another important local time parameter for each subtask which is its local deadline.

Definition 2. A **local deadline** $D_{i,j}$ of subtask $\tau_{i,j}$ is defined as the latest possible deadline of $\tau_{i,j}$ with $D_{i,j} = D_i - R_i(\tau_{i,j})$, where $R_i(\tau_{i,j})$ is the length of the longest execution path from the successor of subtask $\tau_{i,j}$ to the ending subtask in the DAG ($C_{i,j}$ is not included).

For each subtask $\tau_{i,j}$ of DAG τ_i, we calculate a local deadline $D_{i,j}$. As in the case of local offset, the local deadline is calculated while considering the best execution case where we assume a system with an infinite number of processors. In order for a DAG τ_i to respect its deadline D_i, each subtask $\tau_{i,j}$ has to keep enough time for its successors to execute. The local deadline of a subtask can be seen as the longest sequential execution path from its successors to the ending subtasks on a system of infinite processors.

LEMMA 1. *If a subtask $\tau_{i,j}$ misses its local deadline $D_{i,j}$ at time t, the DAG τ_i will definitely miss its deadline even if $t \leq D_i$.*

PROOF. We consider the scenario where the system has an infinite number of processors, the local deadline of a subtask is its latest possible deadline that leaves just enough time for the execution of their successors. So missing this deadline at t means that the time from t to the deadline of the DAG is not enough for the successor subtasks to execute even if the system has an infinite number of processors. □

In Algorithm 2, we showed a straightforward recursive method based on the depth-first search algorithm that calculates the local deadline of each subtask in the DAG. This algorithm executes in linear time.

Algorithm 2 Local deadline algorithm

```
        ▷ Inputs: τ_i is a graph task, τ_{i,j} is a subtask in τ_i
procedure LOCAL_DEADLINE(τ_{i,j})
    if τ_{i,j} = end_subtask(τ_i) then
        return d_i
    else
        return min_{τ_{i,k} ∈ Succ(τ_{i,j})} (
        LOCAL_DEADLINE(τ_{i,k}) - C_{i,k})
    end if
end procedure
```

Back to the example in Figure 3, we calculate the local deadline $D_{1,j}$ for each subtask $\tau_{1,j}$ in DAG τ_1. We start with the ending subtask $\tau_{1,6}$. This subtask has no successor subtask so it has the same deadline as the DAG $D_{1,6} = D_1 = 8$. The predecessor subtasks of $\tau_{1,6}$ are $\tau_{1,2}, \tau_{1,4}, \tau_{1,5}$. These subtasks must have a local deadline that guarantees the schedulability of $\tau_{1,6}$ in the best case. For the three subtasks, their local deadline is equal to $D_{1,6} - C_{1,6}$, and so forth for the rest of the subtasks.

As a result, a new notation can be added to subtask definition to include the local offset and deadline. A constrained-deadline subtask $\tau_{i,j}$ in τ_i is characterized by a three-tuple $\{O_{i,j}, C_{i,j}, D_{i,j}\}$. In our work, these local offsets and deadlines are not used in the scheduling process of DAGs because we assume that the scheduling decisions are taken at DAG level, but they are useful in the analysis. Regarding the period of each subtask, we can notice that each subtask inherits the period of its DAG ($T_{i,j} = T_i$).

In Figure 3(b) we show the time diagram of each subtask in DAG τ_1 individually. It is worth mentioning that this is not a decomposition algorithm and we did not transform the DAG task into another model. Each subtask is still a dependent subtask which has to wait for the completion of its predecessors to complete their execution in order to become ready for execution.

5. G-EDF SCHEDULABILITY TEST

In this paper, we consider Global Earliest-Deadline First (G-EDF) scheduling algorithm for a taskset τ of n sporadic constrained-deadline DAGs on m identical processors. The G-EDF scheduling algorithm is a well-known fixed job priority assignment algorithm, in which, the ready job with the earliest absolute deadline has the highest priority. A global scheduler in multiprocessor systems is defined as the scheduler which assigns active ready jobs on available processors without considering the previous state of the processors. As a result, job migrations and preemptions are allowed.

In this work, we assume that the scheduling decisions are taken at the DAG level and not at the subtask level, which means that the DAG job with the earliest absolute deadline has the highest priority. When G-EDF is used on DAG taskset, a sufficient condition for the schedulability of a particular job was proved by Li et al. in [15] in the following lemma:

LEMMA 2. *(from [15]) If the total workload $A^{k,a}$ on the a^{th} job of DAG task τ_k is bounded by*

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

then this job can meet its deadline on m identical processors with a speed of b.

On a system of m processors of speed b, each time step is divided into b sub-steps. Let a **complete** sub-step be defined as the sub-step where all the m processors of the system are busy, while an **incomplete** sub-step is the sub-step when at least one processor is idle. The proof of Lemma 2 is based on two straightforward observations for the schedulability of DAGs:

- At each incomplete sub-step, the remaining critical path length for each unfinished job is decreased by 1.
- The total work F^t done in an interval of length corresponding to t steps on m processors of speed b, in which there are t^* incomplete sub-steps is defined by the relation:

$$\begin{aligned} F^t &\geq m(bt - t^*) + t^* \\ &\geq bmt - (m-1)t^* \end{aligned}$$

As we can see from the previous bound and observations, no knowledge of internal DAG structure is required. In Lemma 2, an upper bound is defined on the total workload done for a particular job of a DAG task in an interval equal to its deadline, on a system of m processors of speed b. The authors in [15] found that any periodic implicit-deadline taskset of DAGS satisfies the previously mentioned necessary conditions and is schedulable on a system of m processors of speed $b \geq 4 - \frac{2}{m}$ using G-EDF. It's worth mentioning that the right side of the condition in Lemma 2 can be relaxed due to the knowledge of the internal structure of the DAG. However, we will use this upper bound in this work while keeping this as a perspective work for the future.

In this work, we use the work bound expressed in Lemma 2 on a taskset of n sporadic constrained-deadline DAG tasks on m identical processors. Our main contribution is to analyze each DAG in the taskset while including its internal structure to find an upper bound on the work of a particular DAG job. Then we use this bound to find the minimum required speed b of processors which guarantees the schedulability of the taskset using G-EDF. Although adding the knowledge of internal structure might lead to a more complicated analysis, we show that the analysis is more accurate and the bounds are less pessimistic. Therefore, we will consider the local offsets and deadlines of each subtask in the DAG tasks in our analysis.

The total workload done on a particular job of a DAG task in an interval equal to its deadline consists of the execution time of this job and the interference from the higher priority jobs of other DAGs. In order to find an upper bound of workload, first we have to identify the scenario which generates the highest interference on a particular job. Then we analyze the possible interference on DAGs w.r.t. this scenario.

Interference analysis on DAGs

In our work, we assume that each DAG task τ_i in the taskset generates an infinite number of jobs each denoted by J_i, since we consider constrained-deadline DAGs, where the deadline of a DAG cannot exceed its period, only one active job of each DAG is released at any time t. As a result, when G-EDF is used, the interference on a particular job J_i cannot be generated from jobs of DAG τ_i itself but from the other DAGs.

In this case, there are two types of jobs causing interference on DAG job J_i in an interval of length D_i: **body** jobs and **carry-in** jobs. A body job is a job that is released after the release of J_i and has an absolute deadline no later than the absolute deadline of J_i. The carry-in job is a job released before the release of job J_i with its deadline in the interval $(r_i, d_i]$. Figure 4 shows an example of an interfering DAG task which has the two types of interfering jobs.

Worst-case interference scenario for DAGs

Lemma 2 is based on the workload on a particular job of DAG task. In order to find the upper bound, either we have to calculate the workload of each possible job and then choose the maximum, or we can identify the scenario that generates the maximum interference. The *maximum interference* on a DAG task τ_i is defined as the longest cumulative time intervals in D_i in which any subtask of τ_i is ready to execute but it is blocked by higher priority DAG tasks in the system.

OBSERVATION 1. *For sporadic DAG tasks, the maximum total interference on a particular job of DAG τ_i in an interval of length D_i occurs when the jobs of the other DAGs are activated periodically.*

This is a straightforward observation, because the interference of a DAG task τ_j on a particular job of another DAG τ_i depends on the number of jobs within the interfering interval. Sporadic activation means that the time interval between two successive jobs of τ_j is at least T_j. So the maximum number of jobs within a fixed interval happens if the jobs are activated as soon as possible (after exactly T_j time units).

Therefore, in order to define the worst-case interference scenario on a particular job in a sporadic taskset, we consider periodic tasksets in the analysis.

LEMMA 3. *When G-EDF is used and no deadline is missed, the interference of a DAG task τ_j on a particular job of τ_i in an interval of length D_i is maximized when the deadline of the last body job of τ_j is the same as the deadline of τ_i, and the carry-in job of τ_j executes just before its deadline.*

PROOF. The proof of this lemma is inspired from the worst-case interference scenario described in [6] for sequential tasks.

As shown in Figure 4, the interference bound can be easily analyzed considering this situation, since the carry-in job contributes to its maximum interference in the interval. While moving the interval backward or forward will only result in reducing the interference on the job. □

Let J_i^* denotes the job of DAG τ_i which is defined in the scenario in Lemma 3, and let $J_{i,j}^*$ be the job of subtask $\tau_{i,j}$.

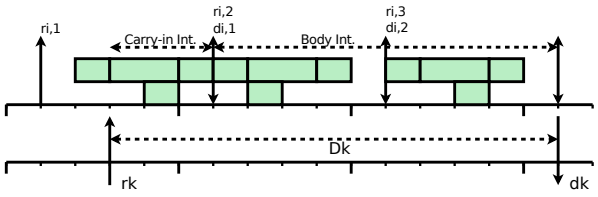


Figure 4: The scenario that generates the worst interference from DAG τ_i on J_k^* of DAG τ_k.

In the DAG model, if a job of DAG τ_i executes just before its deadline D_i (as late as possible), then this means that each subtask $\tau_{i,j}$ will execute just before its calculated local deadline $D_{i,j}$.

Body jobs interference

Based on the definition of a body job, it contributes to its full execution time C_j in the interference on J_i^*. According to this, defining the internal structure of each DAG task does not affect the total interference. In order to calculate this interference, we will use the Demand Bound Function (DBF) introduced in [2]. For constrained-deadline sequential tasks, DBF is defined as follows.

Definition 3. The **Demand Bound Function** (DBF^i) of a sequential task τ_i in a time interval $[0,t]$ for any $t > 0$ is defined as the sum of the execution time of all jobs of τ_i that have both their arrival time and deadline $[0,t]$.

$$DBF^i(t) = \max(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) * C_i)$$

LEMMA 4. *The total body work interference on J_k^* of DAG task τ_k is the sum of the demand bound function of all the DAGs in taskset in the time interval $[0, D_k]$.*

PROOF. In the DAG model, the subtasks in a DAG are sequential tasks with precedence constraints, which have local offsets and deadlines. Let $DBF_k^{i,j}$ denote the demand bound function from the jobs of subtask $\tau_{i,j}$ in $[0, D_k]$:

$$DBF_k^{i,j} = \max(0, \left(\left\lfloor \frac{D_k - D_{i,j}}{T_{i,j}} \right\rfloor + 1\right) * C_{i,j})$$

Since all the subtasks of a DAG have the same period of the DAG ($T_{i,j} = T_i$). The DBF of a DAG τ_i can be calculated as follows:

$$DBF_k^i = \sum_{j=1}^{n_i} \max(0, \left(\left\lfloor \frac{D_k - D_{i,j}}{T_i} \right\rfloor + 1\right) * C_{i,j})$$
$$= \sum_{j=1}^{n_i} DBF_k^{i,j}$$

The proof of the lemma follows. □

Carry-in jobs interference

The carry-in interference is defined as the interference from a DAG job J_i on another job J_k^* of DAG τ_k, in which J_i is released before the release of J_k and has a deadline in the interval $(r_k, r_k + D_k]$. For constrained-deadline tasks, there is, at the most, one carry-in job from each DAG in the taskset.

Let α_k^i denotes the interfering interval of carry-in DAG job J_i on DAG job J_k^* which can be defined as the interval between the release of J_k and the deadline of J_i, where $\alpha_k^i = d_i - r_k$. It is straightforward to define the carry-in interference for sequential tasks, but it is more complicated in the case of parallel tasks of a DAG model. As we can see in the example in Figure 2, the execution flow of subtasks is defined by precedence constraints, that define which subtasks are included in the carry-in interference and which have no effect. Let $\alpha_k^{i,j}$ denote the carry-in interfering interval of subtask $\tau_{i,j}$ on job J_k^*, which is defined as: $\alpha_k^{i,j} = D_{i,j} - r_k$. As shown in Figure 5, subtasks of a DAG job can be divided into the following categories based on their carry-in interference:

- a subtask $\tau_{i,j}$ that has its local deadline $D_{i,j}$ before the release of J_k^*. This subtask causes no carry-in interference, and $\alpha_k^{i,j}$ has a negative value,
- a subtask $\tau_{i,j}$ that has its local offset after the release of J_k^*. This subtask contributes to its full execution time as carry-in interference,
- a subtask $\tau_{i,j}$ is released before the release of J_k^* and has a deadline in the interval of length α_k^i. This subtask has partial carry-in interference on J_k^* equal to $\alpha_k^{i,j}$.

Let ζ_k^i denote the carry-in interference of DAG job τ_i on another DAG job τ_k and let it be defined as:

$$\zeta_k^i = \sum_{\tau_{i,j} \in \tau_i} \min\left(C_{i,j}, \max\left(0, \alpha_k^{i,j}\right)\right) \quad (1)$$

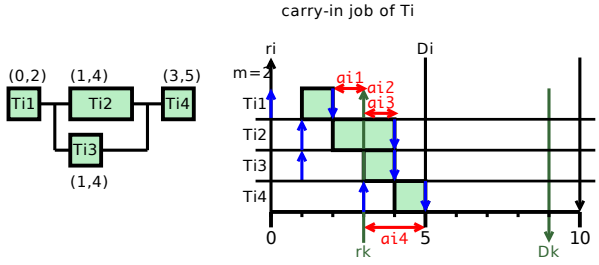


Figure 5: Carry-in interference of subtasks of DAG τ_i on DAG τ_k. The shaded areas show the worst carry-in interference work from each subtask.

LEMMA 5. *When G-EDF is used and no deadline is missed, the carry-in interference ζ_k on job J_k^* from other DAGs in the system in an interval of length D_k is at most:*

$$\zeta_k \leq \sum_{i=1}^{n} \sum_{j=1}^{n_i} \min\left(C_{i,j}, \max\left(0, \alpha_k^{i,j}\right)\right) \quad (2)$$

PROOF. The proof of this lemma is based on the situation described above, in which we note that the worst carry-in interference of a DAG on another one happens when the

carry-in job is executed before its deadline. In this situation, each subtask $\tau_{i,j}$ of the carry-in job cannot contribute to more than the interfering interval on job J_k^* which is of length $\alpha_k^{i,j}$. And since each subtask $\tau_{i,j}$ is sequential, it cannot contribute to more than its WCET in this interval. □

G-EDF scheduling condition

From the previous analysis, we derive the following theorem.

THEOREM 6. *A set of DAG τ is G-EDF schedulable on m processors of speed b if:*

$$\forall k \in \{1, \ldots, n\},$$
$$\sum_{i=1}^{n} DBF_k^i + \sum_{i=1, i\neq k}^{n} \zeta_k^i \leq bmD_k - (m-1)D_k \quad (3)$$

PROOF. The proof of this theorem has already been described in previous sections. The left side of the inequality is an upper bound for the interference on DAGs as proved in Lemma 4 and 5. This upper bound represents the total workload for a particular DAG job during an interval equal to its deadline, which is bounded from Lemma 2 (right side of inequality). □

Here we should mention that the workload analysis is done based on the worst-case interference scenario, in which each subtask has to execute up to its WCET. Hence, the body and carry-in interference are maximized for the interfering subtasks. Based on this theorem, for each DAG task in the taskset, we find the minimum value of b satisfies inequality of Equation (3). Then for all the DAGs, the maximum b is the speed of the m processors that guarantee the schedulability of the taskset when G-EDF is used.

6. SUSTAINABILITY

The schedulability analysis aims at ensuring that a taskset is schedulable according to a scheduling policy when it meets all its deadlines. A necessary requirement is that the scheduling policy be *stable* to "positive" changes of the task parameters. For example, if a taskset with processor utilization is schedulable according to a given scheduling policy, then it must be schedulable with a smaller utilization. Otherwise, we can state that this policy is subject to scheduling anomalies. The sustainability w.r.t. positive variation in parameters has been studied in the case of EDF scheduling on uniform multiprocessors in [5].

In the same way considering the scheduling policy, the notion of sustainability can be applied to schedulability tests. The common FP and EDF tests for uniprocessors have been examined in [4, 7]. Concerning the multiprocessor case, both the scheduling policies and the schedulability tests have been discussed in [1] from the sustainability point of view. In particular, the G-EDF scheduling policy has been shown to be sustainable w.r.t. smaller execution requirements and later arrival times (sporadic case). However, sustainability of G-EDF w.r.t. larger relative deadlines is not so straightforward. Indeed, it depends on the implementation of the G-EDF scheduling policy. If the priorities of jobs are computed using the priorities of tasks which generate these jobs (the *specified* priority), then G-EDF is trivially sustainable w.r.t. larger relative deadlines since a smaller *actual* deadline of a job does not affect the scheduling decisions. Otherwise, it is not obvious that this property is guaranteed and it is safer to design the G-EDF scheduler to compute job priorities according to the specified priorities.

6.1 Scheduling policy

In this section, we review the property of sustainability of G-EDF scheduling policy in the case of tasksets composed of DAGs. Firstly, we give the definition of sustainability according to a scheduling policy. Secondly, we discuss three observations related to the three kind of parameter relaxations.

Definition 4. (from [1]) Let A denote a scheduling policy. Let τ denote any sporadic task system that is A-schedulable. Let J denote a collection of jobs generated by τ. Scheduling policy A is said to be sustainable if and only if A meets all deadlines when scheduling any collection of jobs obtained from J by changing the parameters of one or more individual jobs in any, some, or all of the following ways: (i) decreased execution requirements; (ii) larger relative deadlines; and (iii) later arrival times with the restriction that successive jobs of any task $\tau_i \in \tau$ arrive at least T_i time units apart.

According to this definition, we notice that G-EDF is sustainable w.r.t. the three possible relaxations of the parameters of DAG jobs.

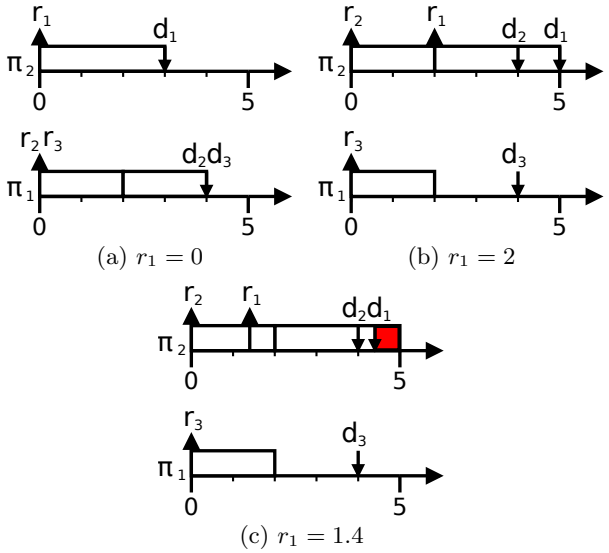


Figure 6: An example of job collection generated by the sporadic $\{\tau_1(3,3), \tau_2(2,4), \tau_3(2,4)\}$ implicit-deadline taskset where τ_i is characterized by (C_i, T_i).

In order to apply the observations from [1], we used the example given in Figure 6. This example represents three different activation schemes that are generated by the same taskset. For the sake of simplicity, we considered simple sequential jobs instead of DAG jobs.

OBSERVATION 2. *The G-EDF scheduling policy is sustainable w.r.t. decreased execution requirements for any set of jobs that may be generated by a set of sporadic DAG.*

PROOF. As for Observation 1 in [1], this proof is based on the proof of predictability in [13, 12]. This predictability proof can be applied easily on a collection of DAG jobs by considering each subtask job as an independent one. □

We observe the sustainability of G-EDF w.r.t. decreased execution requirement by means of Definition 4. We consider a collection of jobs which have been generated by G-EDF and with known release times. If this collection can be accurately scheduled w.r.t. these release times, no deadline is missed by decreasing execution requirements. In a more general case, online G-EDF can generate several collections of jobs from the same sporadic taskset. The same job can have different release times in several collections. We should consider jobs with release jitters if we want to observe the sustainability at taskset level. Unfortunately, it has been proved in [13] that a preemptable, migratable and jittered release timed jobs scheduling is not predictable.

OBSERVATION 3. *The G-EDF scheduling policy is sustainable w.r.t. later arrival times for any set of jobs that may be generated by a set of sporadic DAG.*

PROOF. Let τ denote a sporadic DAG taskset of DAG that is G-EDF schedulable. Let J' denote any collection of jobs obtained from J by increasing the arrival times of one or more individual jobs with the restriction that successive jobs of any task $\tau_i \in \tau$ arrive at least T_i time units apart. The collection J' of jobs could also have been generated by τ since it is G-EDF schedulable. The observation follows, as a consequence. □

According to Figure 6, the consequence could not be straightforward. In this example, we represent three jobs generated by a sporadic taskset. In Figure 6(a) (respectively Figure 6(b)), job J_1 is released at time $t = 0$ (respectively at time $t = 2$) and all jobs meet their deadline. However, in Figure 6(c), J_1 misses its deadline at time $t = 4$ since it is released at time $t = 1.4$ but no processor is available before time $t = 2$. In order to make it clear with the observation, we recall in the proof that the sporadic taskset has to be schedulable according to G-EDF, thus all possible release scenarios must be tested schedulable.

OBSERVATION 4. *The G-EDF scheduling policy is sustainable w.r.t. larger relative deadlines for any set of jobs that may be generated by a set of sporadic DAG if the scheduling algorithm is implemented by using the specified deadlines.*

PROOF. As explained in the section above, this observation is based on the implementation of G-EDF. If the scheduling algorithm uses a larger relative deadline to compute job priorities, it is not clear that G-EDF is sustainable w.r.t. deadline relaxations. But if the algorithm computes the priorities by considering the relative deadline of the tasks, no change in the scheduling behavior will occur. □

The sustainability of G-EDF, w.r.t the parameters shown in Observation 2, 3 and 4, is a good result. Those observations are based on the fact that we considered a sporadic DAG set (or taskset in a more general way) as G-EDF schedulable. We now propose a sufficient feasibility condition.

6.2 Schedulablity test

We have shown that G-EDF is a sustainable scheduling policy for a sporadic DAG set. We now consider the sustainability of the schedulability test proposed in Section 5 according to the following definition.

Definition 5. (from [1]) Let A denote a scheduling policy, and F an A-schedulability test for sporadic task systems. Let τ denote any sporadic task system deemed to be A-schedulable by F. Let J denote a collection of jobs generated by τ. F is said to be a sustainable schedulability test if and only if scheduling policy A meets all deadlines when scheduling any collection of jobs obtained from J by changing the parameters of one or more individual jobs in any, some, or all of the following ways: (i) decreased execution requirements; (ii) larger relative deadlines; and (iii) later arrival times with the restriction that successive jobs of any task $\tau_i \in \tau$ arrive at least T_i time units apart.

OBSERVATION 5. *The test proposed in Section 5 is sustainable for sporadic DAG w.r.t. decreased execution requirements, larger relative deadlines and later arrival times.*

PROOF. A DAG set deemed schedulable by the test proposed in Section 5 is G-EDF schedulable. G-EDF is a sustainable scheduling policy w.r.t. decreased execution requirements, larger relative deadlines and later arrival times. The observation follows. □

In addition to the sustainability of our G-EDF DAG schedulability test, we studied the *self-sustainability* of our test.

Definition 6. (from [1]) A schedulability test is self-sustainable if all task systems with "better" (less constraining) parameters than a task system deemed to be schedulable by the test are also deemed schedulable by the test.

In this case, the taskset is not only required to remain schedulable under various parameter relaxations, but it is required to be verifiably schedulable by the same test.

OBSERVATION 6. *G-EDF DAG schedulability test from Theorem 6 is self-sustainable w.r.t. decreased execution requirements.*

PROOF. In Equation 3, decreased execution requirements can only decrease the left side of the inequality since only DBF and ζ_i^k values are dependent on the WCET values. The inequality remains valid and the observation follows. □

OBSERVATION 7. *G-EDF DAG schedulability test from Theorem 6 is self-sustainable w.r.t. later arrival times.*

PROOF. In Equation 3, later arrival times can only decrease the left side of the inequality since only DBF values are dependent on the period values. The inequality remains valid and the observation follows. □

OBSERVATION 8. *G-EDF DAG schedulability test from Theorem 6 is not self-sustainable w.r.t. larger relative deadlines.*

PROOF. We assume a system with only one unit-speed processor. From Equation (3), we obtain:

$$\sum_{i=1}^{n} DBF_k^i + \sum_{i=1, i\neq k}^{n} \zeta_k^i \leq D_k$$

We recall that ζ_k^i is defined by the following expression:

$$\sum_{\tau_{i,j} \in \tau_i} \min\left(C_{i,j}, \max\left(0, D_{i,j} - r_k\right)\right) \tag{4}$$

Let us assume that in Equation (4), the "min" value is given by $\max(0, D_{i,j} - r_k)$. Then we can assume that there is

$D'_{i,j} > D_{i,j}$ such that $\max(0, D'_{i,j} - r_k) > \max(0, D_{i,j} - r_k)$ but $\max(0, D'_{i,j} - r_k) \leq C_{i,}$. The deadline $D'_{i,j} > D_{i,j}$ implies a larger carry-in interference of a job of τ_i on a job of τ_k which can miss its deadline. □

Unfortunately, our proposed schedulability test for G-EDF is not self-sustainable w.r.t. larger relative deadlines. These results can be explained because the test is based on the analysis of the schedule on a study window of size corresponding to the DAG deadline. A self-sustainable schedulability test w.r.t larger relative deadlines is subject to future work.

7. EXPERIMENTAL RESULTS

We presented in Theorem 6 a scheduling bound based on the total workload of a DAG task during an interval equal to its deadline. This bound can be used as a G-EDF schedulability condition by calculating the processor speed (denoted by b in Theorem 6) that guarantees the schedulability of the taskset. As mentioned earlier, for a particular taskset, the speed of processors can be found by solving the inequality of Equation (3) for each DAG tasks. The minimum processor speed among all the tasks is the speed that guarantees G-EDF schedulability.

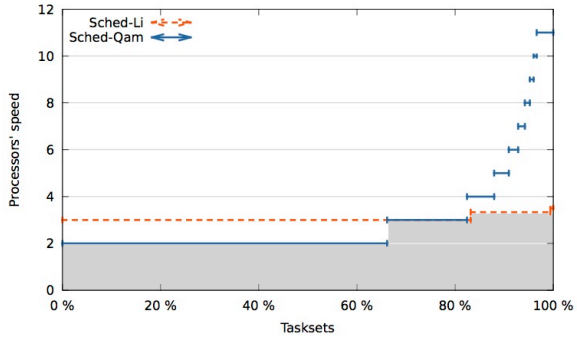
We use simulation to show the performance of our scheduling condition given in theorem 6 for tasksets of DAGs. Then, we compared our calculated processor speed with the speed calculated in [15] ($b \geq 4 - \frac{2}{m}$). We used a simulation tool called *YARTISS* [8], which is an open-source software written in Java. It contains real-time scheduling policies on multiprocessor systems for various models of tasks including DAGs.

Regarding the generation of DAG tasksets, our DAG generator is based on the Uunifast-Discard algorithm [10] for random generation of tasks. This algorithm is proposed by Davis and Burns to generate randomly a set of tasks of a certain total utilization on multiprocessor systems. The number of tasks and their utilization are inputs of this algorithm. The DAG taskset generator is described briefly as follows:

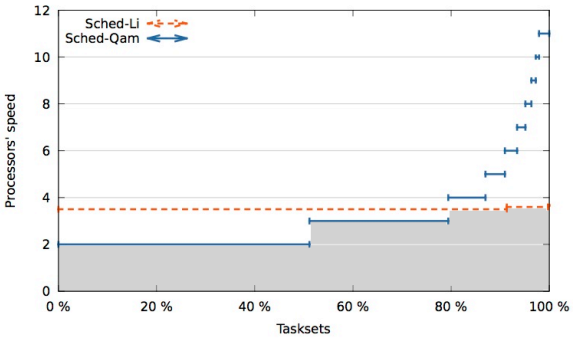
- The algorithm takes two parameters n and U, where n is the number of DAGs in the set and U is the total utilization of the taskset ($U > 0$).
- The Uunifast-Discard algorithm distributes the total utilization on the taskset. A DAG task τ_i can have a utilization U_i greater than 1.
- The subtasks and the directed relations of each DAG are generated randomly based on the calculated utilization. The subtasks are classical sequential real-time tasks.

For our experiments, we fixed the size of the tasksets at 50 periodic DAG tasks per taskset. We chose to simulate periodic DAGs instead of sporadic because the periodic activation generates the worst interference on a particular task in the worst-case scenario. For each utilization from 1 to 5, we generated 1 million tasksets to be simulated on m processors where m is the smallest integer value that is not less than the utilization U of the taskset τ ($m = \lceil U \rceil$).

First, we analyzed the scheduling condition of G-EDF described in Theorem 6. For the sake of clarity, let *Sched-Qam* denote our scheduling condition described in theorem 6, and *Sched-Li* be the condition from [15]. The simulation results for various utilization are shown in Figure 7. More than 80% of the tasksets were schedulable using *Sched-Qam* on m processors with a lower speed than *Sched-Li*. We can see here that *Sched-Qam* calculates processor speed while considering worst-case workload scenario, that is reason why some tasksets required high speed processors. For these 20% tasksets, we can use the workload bound of *Sched-Li* to find the minimum processor speed to guarantee its G-EDF schedulability. As a result, all the tasksets are scheduled using G-EDF on m processors with speed $\leq 4 - \frac{2}{m}$.



(a) Total taskset utilization of 2.



(b) Total taskset utilizatoin of 4.

Figure 7: Processors' speed calculated for each taskset based on Theorem 6.

In the second part of our simulation, we compare the processor speed necessary to schedule the taskset of DAGs using G-EDF when our workload bound and the bound from [15] were used. We generate 5 datasets of utilization from 1 to 5, and each dataset contained 1,000,000 tasksets. After applying *Sched-Qam* and *Sched-Li* on the tasksets, we found that both conditions are not comparable, which means that *Sched-Qam* schedules some tasksets with a smaller speed than *Sched-Li* and vice versa. Despite this result, we found that *Sched-Qam* dominates *Sched-Li* in practice, which means that on average, *Sched-Qam* schedules at least 80% of tasksets on processors with lower speed than *Sched-Li* (tested utilization are 2,4 and 8).

The results of the experimental simulations prove the importance of including the internal structure for the scheduling analysis of DAGs. In average case, our *Sched-Qam* condition had better performance than *Sched-Li*, and schedules tasksets on the same number of processors but with lower speed.

8. CONCLUSION AND FUTURE WORK

In this paper, we were interested in scheduling parallel real-time scheduling on multiprocessor systems, for Directed Acyclic Graph (DAG) tasks. Our motivation was to show that the scheduling of real-time DAG tasks is affected by the internal structure of the DAG and the execution flow of its subtasks. Since the execution of the DAG tasks is not uniform and the subtasks are activated on the basis of the completion time of their successors. In other words, the subtasks of a DAG task have a dynamic activation time.

We analyzed the real-time DAG tasks, and we proposed two algorithms to calculate local offsets and deadlines for each subtask in the DAG. Then we investigated the schedulability of sporadic constrained-deadline DAG tasks on a system of homogeneous processors with Global-EDF. It is based on the workload analysis which includes the interference of subtasks in the analysis. Then we derived a schedulability test for the DAG tasks in which we can calculate in practice the minimum processor speed required to guarantee the schedulability of the DAG set. We studied the sustainability of the scheduling policy and test for the DAG tasks w.r.t. worst-case execution time, period and deadline. We showed the performance of this test by simulations, and we found that on average our schedulability test requires a smaller processor speed than the test in *Li et al.* in [15].

As a future work, we will analyze the schedulability of real-time DAG tasks with other scheduling algorithms than G-EDF, and compare their performance in order to find the most appropriate scheduling algorithms for DAG tasks. We also want to improve our scheduling test and make it sustainable w.r.t. larger relative deadlines.

Acknowledgments

We would like to thank Sanjoy Baruah from the university of North Carolina-Chapel Hill for his interesting comments on this work.

9. REFERENCES

[1] T. P. Baker and S. K. Baruah. Sustainable Multiprocessor Scheduling of Sporadic Task Systems. In *Proceedings of the 21st Euromicro Conference on Real-time Systems (ECRTS)*, pages 141–150, Dublin, Irland, July 2009. IEEE Computer Society.

[2] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190, 1990.

[3] S. K. Baruah, V. Bonifaciy, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A Generalized Parallel Task Model for Recurrent Real-time Processes. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS)*, pages 63–72, San Juan, Puerto Rico, Dec. 2012. IEEE Computer Society.

[4] S. K. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)*, pages 159–168, Rio de Janeiro, Brazil, Dec. 2006. IEEE Computer Society.

[5] S. K. Baruah, S. H. Funk, and J. Goossens. Robustness Results Concerning {EDF} Scheduling upon Uniform Multiprocessors. *IEEE Transactions on Computers*, 52(9):1185–1195, Sept. 2003.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, Apr. 2009.

[7] A. Burns and S. K. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering (JCSE)*, 2(1):74–97, Mar. 2008.

[8] Y. Chandarli, F. Fauberteau, M. Damien, S. Midonnet, and M. Qamhieh. YARTISS: A Tool to Visualize, Test, Compare and Evaluate Real-Time Scheduling Algorithms. In *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Pisa, Italy, 2012.

[9] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[10] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[11] R. I. Davis and B. Alan. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. *ACM Computing surveys*, pages 1 – 44, 2011.

[12] R. Ha. *Validating timing constraints in multiprocessor and distributed real-time systems*. PhD thesis, University of Illinois, Dept. of Computer Science, Urbana-Champaign, IL, USA, 1995.

[13] R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS)*, pages 162–171, Pozman, Poland, June 1994. IEEE Computer Society.

[14] K. Lakshmanan, S. Kato, and R. (Raj) Rajkumar. Scheduling Parallel Real-Time Tasks on Multi-core Processors. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS)*, pages 259–268, San Diego, CA, USA, 2010. IEEE Computer Society.

[15] J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of Global {EDF} for Parallel Tasks. In *Proceedings of the 25th Euromicro Conference on Real-time Systems (ECRTS)*, pages 3–13, Paris, France, 2013. IEEE Computer Society.

[16] C. Liu and J. Anderson. Supporting soft real-time parallel applications on multicore processors. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, pages 114–123, 2012.

[17] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core Real-time Scheduling for Generalized Parallel Task Models. In *The 32nd IEEE Real-Time Systems Symposium (RTSS 2011)*, pages 217–226, 2011.

[18] sebastien Collette, L. Cucu, and J. Goossens. Integrating Job Parallelism in Real-Time Scheduling Theory. *Information Processing Letters*, 106:180–187, 2008.