Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited parallelism

Sébastien Collette\(^\ast\)  Liliana Cucu\(^\dagger\)  Joël Goossens

Université Libre de Bruxelles, C.P. 212
50 Avenue Franklin D. Roosevelt
1050 Brussels, Belgium

E-mail: \{sebastien.collette, liliana.cucu, joel.goossens\}@ulb.ac.be

Abstract

We investigate the global scheduling of sporadic, implicit deadline, real-time task systems on identical multiprocessor platforms. We provide a task model which integrates work-limited job parallelism. For work-limited parallelism, we prove that the time-complexity of deciding if a task set is feasible is linear relatively to the number of (sporadic) tasks for a fixed number of processors. Based on this proof, we propose an optimal scheduling algorithm. Moreover, we provide an exact feasibility utilization bound.

1 Introduction

The use of computers to control safety-critical real-time functions has increased rapidly over the past few years. As a consequence, real-time systems — computer systems where the correctness of each computation depends on both the logical results of the computation and the time at which these results are produced — have become the focus of much study. Since the concept of “time” is of such importance in real-time application systems, and since these systems typically involve the sharing of one or more resources among various contending processes, the concept of scheduling is integral to real-time system design and analysis. Scheduling theory as it pertains to a finite set of requests for resources is a well-researched topic. However, requests in real-time environment are often of a recurring nature. Such systems are typically modeled as finite collections of simple, highly repetitive tasks, each of which generates jobs in a very predictable manner. These jobs have bounds upon their worst-case execution requirements and their periods, and associated deadlines. In this work, we consider sporadic task systems, i.e., where there are at least \(T_i\) time units between two consecutive instants when a sporadic task \(\tau_i\) generates jobs and the jobs must be executed for at most \(C_i\) time units and completed by their relative deadline \(D_i\).

A particular case of sporadic tasks are the periodic tasks for which the period is the exact temporal separation between the arrival of two successive jobs generated by the task. We shall distinguish between implicit deadline systems where \(D_i = T_i, \forall i\); constrained deadline systems where \(D_i \leq T_i, \forall i\); and arbitrary deadline systems where there is no constraint between the deadline and the period.

The scheduling algorithm determines which job[s] should be executed at each time instant. We distinguish between off-line and on-line schedulers. On-line schedulers construct the schedule during the execution of the system; while off-line schedulers mimic during the execution of the system a precomputed schedule (off-line). Remark that if a task is not active at a given time instant and the off-line schedule planned to execute that task on a processor, the latter is simply idled (or used for a non-critical task).

When there is at least one schedule satisfying all constraints of the system, the system is said to be feasible. More formal definitions of these notions are given in Section 2.

Uniprocessor sporadic (and periodic) real-time systems are well studied since the seminal paper of Liu and Layland [9] which introduces a model of implicit deadline systems. For uniprocessor systems we know that the worst-case arrival pattern for sporadic tasks corresponds to the one of (synchronous and) periodic tasks (see, e.g., [11]). Consequently, the results obtained for periodic tasks apply to sporadic ones as well. Unfortunately, this is not the case upon multiprocessors due to scheduling anomalies (see, e.g., [1]).

The literature considering scheduling algorithms and feasibility tests for uniprocessor scheduling is tremendous. In contrast for multiprocessor parallel machines the problem of meeting timing constraints is a relatively new research area.

Related research. Even if the multiprocessor scheduling of sporadic task systems is a new research field, important results have already been obtained. See, e.g., [2] for a good presentation of these results. All these works con-
sider models of tasks where job parallelism is forbidden (i.e., job correspond to a sequential code). This restriction is natural for the uniprocessor scheduling since only one processor is available at any time instant even if we deal with parallel algorithms. Nowadays, the use of parallel computing is growing (see, e.g., [8]); moreover, parallel programs can be easily designed using the Message Passing Interface (MPI [5, 6]) or the Parallel Virtual Machine (PVM [12, 4]) paradigms. Even better, sequential programs can be parallelized using tools like OpenMP (see [3] for details). Therefore for the multiprocessor case we should be able to describe jobs that may be executed on different processors at the same time instant. For instance, we find such requirements in real-time applications such as robot arm dynamics [13], where the computation of dynamics and the solution of a linear systems are both parallelizable and contain real-time constraints.

When a job may be executed on different processors at the very same instant we say that the job parallelism is allowed. For a task \( \tau_i \) and \( m \) identical processors we define a \( m \)-tuple of real numbers \( \Gamma_i = (\gamma_{i,1}, \ldots, \gamma_{i,m}) \) with the interpretation that a job of \( \tau_i \) that executes for \( t \) time units on \( j \) processors completes \( \gamma_{i,j} \times t \) units of execution. Full parallelism, which corresponds to the case where \( \Gamma_i = (1, 2, \ldots, m) \) is not realistic; moreover, if full parallelism is allowed the multiprocessor scheduling problem is equivalent to the uniprocessor one (by considering, e.g., a processor \( m \) times faster).

In this work, we consider work-limited job parallelism with the following definition:

**Definition 1 (work-limited parallelism)** The job parallelism is said to be work-limited if and only if for all \( \Gamma_i \) we have:

\[
\forall 1 \leq i \leq n, \forall 1 \leq j < j' \leq m, \frac{\gamma_{i,j'}}{\gamma_{i,j}} > \frac{\gamma_{i,j'}}{\gamma_{i,j}}.
\]

For example, the \( m \)-tuple \( \Gamma_i = (1.0, 1.1, 1.2, 1.3, 4.9) \) is not a work-limited job parallelism, since \( \gamma_{i,5} = 4.9 > 1.3 \times \frac{4}{3} = 1.625 \).

Remark that work-limited parallelism requires that for each task (say \( \tau_i \)), the quantities \( \gamma_{i,j} \) are distinct (\( \gamma_{i,1} < \gamma_{i,2} < \gamma_{i,3} < \cdots \)).

The work-limited parallelism restriction may at first seem strong, but it is in fact intuitive: we require that parallelism cannot be achieved for free, and that even if adding one processor decreases the time to finish a parallel job, a parallel job on \( j' \) processors will never run \( j' / j \) times as fast as on \( j \) processors. Many applications fit in this model, as the increase of parallelism often requires more time to synchronize and to exchange data between the parallel processes.

Few models and results in the literature concern real-time systems taking into account job parallelism. Manimaran et al. in [10] consider the non-preemptive EDF scheduling of periodic tasks, moreover they consider that the degree of parallelism of each task is static. Meanwhile, their task model and parallelism restriction (i.e., the sub-linear speedup) is quite similar to our model and our parallelism restriction (work-limited). Han et al. in [7] considered the scheduling of a (finite) set of real-time jobs allowing job parallelism. Their scheduling problem is quite different than ours, moreover they do not provide a real model to take into account the parallelism. This manuscript concerns the scheduling of preemptive real-time sporadic tasks upon multiprocessors which take into account the job parallelism. From the best of our knowledge there is no such result and this manuscript provides a model, a first feasibility test and a first exact utilization bound for such kind of systems. **This research.** In this paper we deal with global scheduling\(^1\) of implicit deadline sporadic task systems with work-limited job parallelism upon identical parallel machines, i.e., where all the processors are identical in the sense that they have the same computing power. We formally define our model, and consider the feasibility problem of these systems, taking into account work-limited job parallelism. For work-limited job parallelism we prove that the feasibility problem is linear relatively to the number of tasks for a fixed number of processors. We provide a scheduling algorithm.

**Organization.** This paper is organized as follows. In Section 2, we introduce our model of computation. In Section 3, we present the main result for the feasibility problem of implicit deadline sporadic task systems with work-limited job parallelism upon identical parallel machines when global scheduling is used. We prove that the feasibility problem is linear relatively to the number of tasks when the number of processors is fixed. In Section 4, we provide a linear scheduling algorithm which is proved optimal. We conclude and we give some hints for future work in Section 5.

## 2 Definitions and assumptions

We consider the scheduling of sporadic task systems on \( n \) identical processors \( \{p_1, p_2, \ldots, p_n\} \). A task system \( \tau \) is composed by \( n \) sporadic tasks \( \tau_1, \tau_2, \ldots, \tau_n \), each task is characterized by a period (and implicit deadline) \( T_i \), a worst-case execution time \( C_i \) and a \( m \)-tuple \( \Gamma_i = (\gamma_{i,1}, \gamma_{i,2}, \ldots, \gamma_{i,m}) \) to describe the job parallelism.

We assume that \( \gamma_{i,0} \overset{\text{def}}{=} 0 \) (\( \forall i \)) in the following. A job of a task can be scheduled at the very same instant on different processors. In order to define the degree of parallelization of each task \( \tau_i \) we define the execution ratios \( \gamma_{i,j}, \forall j \in \{1, 2, \ldots, m\} \) associated to each task-index of processor pair. A job that executes for \( t \) time units on \( j \) processors completes \( \gamma_{i,j} \times t \) units of execution. In this paper we consider work-limited parallelism as given by Definition 1.

We will use the notation \( \gamma_{i,j} \overset{\text{def}}{=} (C_i, T_i, \Gamma_i), \forall i \) with \( \Gamma_i = (\gamma_{i,1}, \gamma_{i,2}, \ldots, \gamma_{i,m}) \) with \( \gamma_{i,1} < \gamma_{i,2} < \cdots < \gamma_{i,m} \).

\(^1\)Job migration and preemption are allowed.
Such a sporadic task generates an infinite sequence of jobs with $T_i$ being a lower bound on the separation between two consecutive arrivals, having a worst-case execution requirement of $C_i$ units, and an implicit relative hard deadline $T_i$. We denote the utilization of $\tau_i$ by $u_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$.

In our model, the period and the worst-case execution time are integers.

A task system $\tau$ is said to be feasible upon a multiprocessor platform if under all possible scenarios of arrivals there exists at least one schedule in which all tasks meet their deadlines.

**Minimal required number of processors.** Notice that a task $\tau_i$ requires more than $k$ processors simultaneously if $u_i > \gamma_i k$; we denote by $k_i$ the largest such $k$ (meaning that $k_i$ is the smallest number such that the task system $\{\tau_i\}$ is feasible on $k_i + 1$ processors):

$$k_i \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } u_i \leq \gamma_i, \\ \max_{k=1}^{\infty} \{k \mid \gamma_i k < u_i\}, & \text{otherwise}. \end{cases}$$

Notice that if $k_i = m$ for any $i$, the task system is infeasible as at least one task requires more than $m$ processors.

For example, let us consider the task system $\tau = \{\tau_1, \tau_2\}$ to be scheduled on three processors. We have $\tau_1 = (6, 4, \Gamma_1)$ with $\Gamma_1 = (1.0, 1, 2, 2.0)$ and $\tau_2 = (3, 4, \Gamma_2)$ with $\Gamma_2 = (1, 0, 1, 2, 1.3)$. Notice that the system is infeasible if the job parallelism is not allowed since $\tau_1$ will never meet its deadline unless it is scheduled on at least two processors. There is a feasible schedule if the task $\tau_1$ is scheduled on two processors and $\tau_2$ on a third one.

**Definition 2 (schedule $\sigma$)** For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of $m$ processors $\{p_1, \ldots, p_m\}$ we define the schedule $\sigma(t)$ of system $\tau$ at instant $t$ as $\sigma : \mathbb{R}_+ \to \{0, 1, \ldots, n\}^m$ where $\sigma(t) \stackrel{\text{def}}{=} (\sigma_1(t), \sigma_2(t), \ldots, \sigma_m(t))$ with

$$\sigma_j(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if there is no task scheduled on } p_j \text{ at instant } t; \\ i, & \text{if } \tau_i \text{ is scheduled on } p_j \text{ at instant } t \end{cases}$$

for all $1 \leq j \leq m$.

**Definition 3 (canonical schedule)** For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of $m$ processors $\{p_1, \ldots, p_m\}$, a schedule $\sigma$ is canonical if and only if the following equations are satisfied:

$$\forall j \in [1, m], \forall t, t' \in [0, 1), t < t' : \sigma_j(t') \leq \sigma_j(t)$$

$$\forall j, j' \in [1, m], j < j', \forall t, t' \in [0, 1) : \sigma_j(t) \leq \sigma_{j'}(t')$$

and the schedule $\sigma$ contains a pattern that is repeated every unit of time, i.e.,

$$\forall t \in \mathbb{R}_+, \forall 1 \leq j \leq m : \sigma_j(t) = \sigma_j(t + 1).$$

Without loss of generality for the feasibility problem, we consider a feasibility interval of length 1. Notice that the following results can be generalized to consider any interval of length $\ell$, as long as $\ell$ divides entirely the period of every task.

### 3 Our feasibility problem

In this section we prove that if a task system $\tau$ is feasible, then there exists a canonical schedule in which all tasks meet their deadlines. We give an algorithm which, given any task system, constructs a canonical schedule or answers that no schedule exists. The algorithm runs in $O(n)$ time with $n$ the number of tasks in the system.

We start with a generic necessary condition for schedulability using work-limited parallelism:

**Theorem 1** In the work-limited parallelism model and using an off-line scheduling algorithm, a necessary condition for a sporadic task system $\tau$ to be feasible on $m$ processors is given by:

$$\sum_{i=1}^{n} (k_i + \frac{u_i - \gamma_i k_i}{\gamma_i k_{i+1} - \gamma_i k_i}) \leq m$$

**Proof.** As $\sigma$ is feasible on $m$ processors, there exists a schedule $\tau$ meeting every deadline. We consider any time interval $[t, t + P]$ with $P \stackrel{\text{def}}{=} \text{lcm}\{T_1, T_2, \ldots, T_n\}$.

Let $a_{i,j}$ denote the duration where jobs of a task $\tau_i$ are assigned to $j$ processors on the interval $[t, t + P]$ using the schedule $\sigma$. $\sum_{j=1}^{m} \sum_{i=1}^{n} j \cdot a_{i,j}$ gives the total processor use of the task $\tau_i$ on the interval (total number of time units for which a processor has been assigned to $\tau_i$). As we can use at most $m$ processors concurrently, we know that

$$\sum_{i=1}^{n} \sum_{j=1}^{m} j \cdot a_{i,j} \leq m \cdot P$$

otherwise the jobs are assigned to more than $m$ processors on the interval. If on some interval of length $\ell$, $\tau_i$ is assigned to $j$ processors, we can achieve the same quantity of work on $j' > j$ processors on an interval of length $\ell \cdot \frac{\gamma_i k_{j'}}{\gamma_i k_j}$. In the first case, the processor use of the task $\tau_i$ is $\ell j$, while in the second case it is $\ell j' \cdot \frac{\gamma_i k_{j'}}{\gamma_i k_j}$. By the restriction that we enforced on the tuple $\Gamma_i$ (see Definition 1), we have

$$\ell j' \cdot \frac{\gamma_i k_{j'}}{\gamma_i k_j} > \ell j$$

Let $\sigma'$ be a slightly modified schedule compared to $\sigma$, where $\forall i \neq i', \forall j, a'_{i,j} = a_{i,j}$. For the task $\tau_i$, it is scheduled on $j'$ processors instead of $j < j'$ in $\sigma$ for some interval of length $\ell$, i.e.
Given any feasible task system \( \tau \), there exists a canonical schedule \( \sigma \) in which all tasks meet their deadlines.

**Proof.** The proof consists of three parts: we first give an algorithm which constructs a schedule \( \sigma \) for \( \tau \), then we prove that \( \sigma \) is canonical, and we finish by showing that the tasks meet their deadline if \( \tau \) is feasible.

The algorithm works as follows: we consider sequentially every task \( \tau_i \), with \( i = n, n-1, \ldots, 1 \), and define the schedule for these tasks in the time interval \([0, 1]\), which is then repeated.

We calculate the duration (time interval) for which a task \( \tau_i \) uses \( k_i + 1 \) processors. If we denote by \( \ell_i \) the duration that the task \( \tau_i \) spends on \( k_i + 1 \) processors, then we obtain the following equation:

\[
\ell_i \gamma_{i,k_i} + (1 - \ell_i) \gamma_{i,k_i} = u_i.
\]

Then, for that task \( \tau_i \),

\[
\sum_{j=1}^m j \cdot a_{i,j} \geq k_i \cdot P + \frac{u_i - \gamma_{i,k_i}}{\gamma_{i,k_i+1} - \gamma_{i,k_i}} \cdot P
\]
and thus

\[
\sum_{i=1}^n \sum_{j=1}^m j \cdot a_{i,j} \leq m \cdot P
\]

\[
\sum_{i=1}^n \left( k_i \cdot P + \frac{u_i - \gamma_{i,k_i}}{\gamma_{i,k_i+1} - \gamma_{i,k_i}} \cdot P \right) \leq m \cdot P
\]

\[
\sum_{i=1}^n \left( k_i + \frac{u_i - \gamma_{i,k_i}}{\gamma_{i,k_i+1} - \gamma_{i,k_i}} \right) \leq m
\]

which is the claim of our theorem.

**Theorem 2** Given any feasible task system \( \tau \), there exists a canonical schedule \( \sigma \) in which all tasks meet their deadlines.

Therefore we assign a task \( \tau_i \) to \( k_i + 1 \) processors for a duration of

\[
\frac{u_i - \gamma_{i,k_i}}{\gamma_{i,k_i+1} - \gamma_{i,k_i}} P
\]
and to \( k_i \) processors for the remainder of the interval, which ensures that the task satisfies its deadline, since each job generated by the sporadic task \( \tau_i \) which arrives at time \( t \) receives in the time interval \([t, t + T_i]\) exactly \( T_i \times u_i = C_i \) time units.

The task \( \tau_n \) is assigned to the processors \((p_m, \ldots, p_{m-k_n})\) (see Figure 1). If \( u_n \neq \gamma_{n,k_n+1} \), another task can be scheduled at the end of the interval on the processor \( p_{m-k_n} \), as \( \tau_n \) does not require \( k_n + 1 \) processors on the whole interval.

![Figure 1. Schedule obtained after scheduling the task \( \tau_n \)](image)

We continue to assign greedily every task \( \tau_i \), by first considering the processors with highest number.

The schedule produced by the above algorithm is canonical as it respects the three constraints of the definition:

- on every processor \( j \) we assign tasks by decreasing index, thus \( \sigma_j(t) \) is monotone and decreasing;
- for all \( i < i' \), if \( \tau_{i'} \) is scheduled on a processor \( p_{j'} \), then \( \tau_i \) is assigned to a processor \( p_j \) with \( j \leq j' \);
- the schedule is repeated every unit of time.

The last step is to prove that if our algorithm fails to construct a schedule, i.e., if at some point we run out of processors while there are still tasks to assign, then the system is infeasible.

In the case of a canonical schedule, \( \lambda_i \) corresponds to:

\[
\lambda_i = k_i + \frac{u_i - \gamma_{i,k_i}}{\gamma_{i,k_i+1} - \gamma_{i,k_i}}.
\]

So for instance, if a task \( \tau_i \) is assigned to \( \lambda_i = 2.75 \) processors, it means that it is scheduled on two processors.
for 0.25 time unit in any time interval of length 1, and on three processors for 0.75 time unit in the same interval.

If our algorithm fails, it means that \( \sum_{i=1}^{n} \lambda_i > m \), which by Theorem 1 implies that the system is infeasible.

\[ \sum_{i=1}^{n} \left( k_i + \frac{u_i - \gamma_{i,k_i}}{\gamma_{i,k_i+1} - \gamma_{i,k_i}} \right) \leq m \]

Please notice that Corollary 3 can be seen as feasibility utilization bound and in particular a generalization of the bound for uniprocessor (see [9]) where a sporadic and implicit deadline task system is feasible if and only if \( \sum_{i=1}^{n} u_i \leq 1 \). Like the EDF optimality for sporadic implicit deadline tasks is based on the fact that \( \sum_{i=1}^{n} u_i \leq 1 \) is a sufficient condition, we prove the optimality of the canonical schedule based on the fact that \( \sum_{i=1}^{n} k_i + \frac{u_i - \gamma_{i,k_i}}{\gamma_{i,k_i+1} - \gamma_{i,k_i}} \leq m \) is a sufficient condition.

**Corollary 4** There exists an algorithm which, given any task system, constructs a canonical schedule or answers that no schedule exists in \( O(n) \) time.

**Proof.** We know that the algorithm exists as it was used in the proof of Theorem 2. For every task, we have to compute the number of processors required (in \( O(1) \) time, as the number of processors \( m \) is fixed), and for every corresponding processor \( j \), define \( \sigma_j(t) \) appropriately. In total, \( O(n) \) time is required.

## 4 Scheduling algorithm

In this section we give a detailed description of the scheduling algorithm provided in the proof of Theorem 2. Based on the results of Section 3 this algorithm is optimal and runs in \( O(n) \) time (see Corollary 4).

For example for the task system \( \tau = \{ \tau_1, \tau_2 \} \) given before we have \( k_1 = 1 \) and \( k_2 = 0 \). By using Algorithm 1 we obtain:

\[
\begin{align*}
\sigma_3(t) &= 2, \forall t \in [0, 0.75) \\
\sigma_3(t) &= 1, \forall t \in [0.75, 1) \\
\sigma_2(t) &= 1, \forall t \in [0, 1) \\
\sigma_1(t) &= 1, \forall t \in [0, 0.75) \\
\sigma_1(t) &= 0, \forall t \in [0.75, 1).
\end{align*}
\]

Notice that in Algorithm 1, we do not consider the optimization relatively to the number of preemptions or migrations and the scheduling algorithm does not provide satisfactory schedules for problems for which this is an issue. Nevertheless, the algorithm can decide the feasibility of every task system.
5 Discussions

Job parallelism vs. task parallelism. In this manuscript we study multiprocessor systems where job parallelism is allowed. We would like to distinguish between two kinds of parallelism, but first the definitions: task parallelism allows each task to be executed on several processors at the same time, while job parallelism allows each job to be executed on several processors at the same time. If we consider constrained (or implicit) deadline systems task parallelism is not possible. For arbitrary deadline systems, where several jobs of the same task can be active at the same time, the distinction makes sense. Task parallelism allows the various active jobs of the same task to be executed on a different (but unique) processor while job parallelism allows each job to be executed on several processors at the same time.

Optimality and future work. In this paper we study the feasibility problem of implicit deadline sporadic task systems with work-limited job parallelism upon identical parallel machines when global scheduling is used. We prove that our problem has a time-complexity that is linear relative to the number of tasks. We provide an optimal scheduling algorithm that runs in $O(n)$ time and we give an exact feasibility utilization bound.

Our algorithm is optimal in terms of the number of processors used. It is left open whether there exists an optimal algorithm in terms of the number of preemptions and migrations. As a first step, we used an interval of length 1 to study the feasibility problem. If we generalize our algorithm to work on an interval of length equal to the gcd of the periods of every task, we decrease the preemptions and migrations. We do not know, however, if the result is optimal.

The definitions of work-limited job parallelism was given here for identical processors, one should investigate an extension of this definition to the uniform processor case.

Acknowledgments

The authors would like to thank Sanjoy Baruah for posing the feasibility problem and Jean Cardinal for taking part in interesting discussions. Finally the detailed comments of an anonymous referee greatly helped in improving the presentation of the manuscript.

References