RESEARCH-ARTICLE

Characterizing G-EDF scheduling tardiness with uniform instances on multiprocessors

GIOVANNI BUZZEGA, University of Modena and Reggio Emilia, Modena, MO, Italy

GIANLUCA NOCETTI, University of Modena and Reggio Emilia, Modena, MO, Italy

MANUELA MONTANGERO, University of Modena and Reggio Emilia, Modena, MO, Italy

Published: 07 June 2023

RTNS 2023: The 31st International Conference on Real-Time Networks and Systems
June 7 - 8, 2023
Dortmund, Germany

RTNS '23: Proceedings of the 31st International Conference on Real-Time Networks and Systems (June 2023)
https://doi.org/10.1145/3575757.3593641
ISBN: 9781450399838

Characterizing G-EDF scheduling tardiness with uniform instances on multiprocessors

Giovanni Buzzega
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Univ. Modena e Reggio Emilia
Modena, Italy
giovanni.buzzega@unimore.it

Gianluca Nocetti
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Univ. Modena e Reggio Emilia
Modena, Italy
246259@studenti.unimore.it

Manuela Montangero
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Univ. Modena e Reggio Emilia
Modena, Italy
manuela.montangero@unimore.it

ABSTRACT

Soft real-time multiprocessor systems frequently adopt the G-EDF (Global-Earliest Deadline First) scheduling policy as it is lightweight and it guarantees bounded tardiness. Much effort has been spent in literature to provide efficiently computable tardiness bounds for periodic task systems scheduled on multiprocessors, but still, no exact bound is known and the best-known approximation takes non-polynomial time to compute.

In this paper, we consider synchronous and periodic task systems in which tasks have the same period length and the same job length, named uniform scheduling instances. We analytically derive a tight bound on the maximum tardiness and provide the exact values of the tardiness of each processor, showing that the latter can be computed in time linear in the number of processors. This result provides a lower bound to tardiness for the more general class of instances and is intended to close the gap with the upper bound from below. We also show that the results can be applied in practice to a wider set of instances than those considered in the theoretical study.

CCS CONCEPTS

• **Theory of computation → Scheduling algorithms**; • **Computer systems organization → Real-time systems**.

KEYWORDS

G-EDF scheduling, Soft Real-Time, Tardiness.

ACM Reference Format:
Giovanni Buzzega, Gianluca Nocetti, and Manuela Montangero. 2023. Characterizing G-EDF scheduling tardiness with uniform instances on multiprocessors. In *The 31st International Conference on Real-Time Networks and Systems (RTNS 2023), June 07–08, 2023, Dortmund, Germany*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3575757.3593641

1 INTRODUCTION

A *Soft Real-Time* (SRT) system must satisfy response time constraints, but unlike what happens with *Hard Real-Time* systems, failing to meet a deadline is not considered a complete system failure. On the contrary, as long as the system tasks are timely executed their results continue to have value, even if the system performance/quality might degrade (the more deadlines are missed, the more the performance/quality is degraded) [8].

The choice of the scheduling policy for a task system depends on a number of factors. In particular, when dealing with SRT applications the policy should introduce minimum overhead (i.e., the time needed to decide which available processor will execute the next pending job should be as negligible as possible) so that all the computation time can be used for job execution. For the same reason, a common choice is to not allow preemption and avoid the overhead introduced by context switching. The *Global Earliest Deadline First* (G-EDF) policy is a common choice for SRT because it is simple to implement and it is low-overheaded [9]. The feasibility of applications scheduled with a given policy depends on the tardiness incurred by instances of the application with that policy, where tardiness is the maximum delay incurred by the system in the completion of a job with respect to its deadline. If tardiness is unacceptable, then the application is considered unfeasible. For SRT applications, tardiness is related to the performance/quality of the system and is tolerated up to some threshold that depends on the application. For these reasons, determining tight bounds for scheduling tardiness is crucial: a bound too loose may cause a feasible SRT application to be mistakenly deemed unfeasible.

In this paper, we consider a special case of *Harmonic Task systems*, i.e. the class of periodic task systems where every period is a multiple of each smaller period [1]. In particular, we consider the case in which all task periods and job lengths are equal, tasks are synchronous, and they are to be scheduled on a multiprocessor with identical unit-speed processors using the G-EDF scheduling policy, without preemption. We refer to such instances as *uniform instances*, to reflect their regularity.

Even though some applications might naturally give rise to uniform instances, these are particularly interesting when security issues [3] are taken into consideration. In such contexts, non-uniform instances are sometimes intentionally turned into uniform ones. Indeed, uniform periods and job lengths make tasks indistinguishable for a malicious external observer, that is then not able to easily gather information about the arrival of specific jobs that are targets of attacks. Moreover, uniform job lengths preclude the possibility to append malicious software at the end of a job.

Much effort has been placed to provide tight bounds to the tardiness of periodic task systems [1, 4, 6, 16], however for the G-EDF scheduling policy no exact bound is known: the best known upper

RTNS 2023, June 07–08, 2023, Dortmund, Germany

ACM ISBN 978-1-4503-9983-8/23/06…$15.00
https://doi.org/10.1145/3575757.3593641

bound is probably NP-hard to compute and computable by means of a Branch&Bound algorithm [11].

Here we present our results concerning the study of tardiness for the set of uniform instances. We are able to exactly characterize the tardiness of the system and the one on each processor. Our main result states that tardiness is always bounded by the job length, and that this is a tight bound, as there are instances for which tardiness is equal to the job length minus one time unit. This result improves the bound given by [1] (that tardiness is not greater than the length of the period) for what concerns instances studied in this paper. Moreover, we present an instance classification that allows us to determine, for each instance, a finite set of values that contains all and only possible tardiness values reached during execution. It turns out that enumerating the values in this set is linear in the minimum among the number of processors and the job length. As a consequence, the exact tardiness value of each instance can be computed as the maximum value of this set. Moreover, we show that these results can be extended to other instances with no uniform job length, but satisfying a given condition on utilization.

The contribution of this paper is twofold. From a theoretical point of view, the tight bound to the tardiness of uniform instances scheduled using the G-EDF policy provides a lower bound to tardiness in general. This result contributes to the analytic research of tardiness bounds, by closing the gap from below. From a practical point of view, we provide a procedure to compute the exact tardiness of any uniform instance of a harmonic task system that takes negligible time in practice. This allows us to determine the feasibility of any instance on the fly, eliminating the need to determine if an application is feasible *a-priori*.

We highlight that, even if we are addressing regular uniform instances with a neat graphical representation, grasping the underneath regularity is not a trivial task. By looking at scheduled instances, it is straightforward to observe that the system has a cyclic behavior in assigning jobs to processors. However, formally proving that the behavior is always cyclic and characterizing the cycle period is a more demanding matter. In particular, the proof of the main general result is long and involves induction with five intertwined conditions. Thus, due to space constraints, in this paper we will present our results, but we are not able to give the formal proof of the main result on tardiness in general. We will show that the result holds for one class of instances called Lambda class and we will present proof for processor tardiness for the other classes. A preliminary characterization of the Lambda class has been given in one of the authors' Bachelor Thesis [14], and a preliminary version of the characterization of general instances and relative proofs have been first presented in one of the authors' Master Thesis [2]. Proofs missing in this paper will be given in its extended version.

Finally, in this paper we will not present simulation results to consolidate our results because we analytically proved exact bounds on tardiness and simulations can not possibly add any other interesting contribution. However, we wish to report that we have implemented a tool to simulate the outcome of using the G-EDF scheduling policy on uniform instances and to give a graphical representation of this output. The tool has been extensively used while studying the problem, and to draw the figures in this paper. It is worth adding that all examples that have been produced confirm the theoretical results presented in this paper, as it should be.

Roadmap. The rest of the paper is organized as follows: in Section 2 we briefly review literature related to our problem; in Section 3 we formally introduce the problem addressed in the paper; in Section 4 we present some preliminary results that characterize uniform instances and that will be used further in the paper; in Section 5 we present our main results about tardiness characterization for uniform instances; in Section 6 we formally prove that the results hold for a specific class of uniform instances; in Section 7 we give a formal proof of our results on processor tardiness for a generic uniform instance. Finally, in Section 8 we conclude the paper with some final remarks and a discussion on future work.

2 RELATED WORK

Regarding tardiness bounds on G-EDF policy in SRT system, both preemptive and non-preemptive G-EDF was first shown to be SRT-optimal on identical multiprocessor platforms in [4], meaning that it is possible to schedule any task set that does not over-utilize the computing time available while keeping the tardiness bounded. These bounds have been improved for preemptive G-EDF by [6] using a technique called *compliant vector analysis* and later extended in [7] for arbitrary sporadic task systems (i.e., cases where the relative deadline of a task may be different to its period parameter). SRT-optimality was later shown to affect a larger class of global algorithms [13].

In 2016 Valente introduced an upper bound called *harmonic bound* [16] that was proved to be up to 30% tighter than its predecessors. Unfortunately, its high computational complexity undermines its applicability; a *Branch and Bound* algorithm for the harmonic bound was developed in [10] and later parallelized in [11].

More research was concerned with studying particular cases of the general problem to obtain tighter tardiness bounds. For example, [12] introduced upper bounds for the *First-in First-out* (FIFO) scheduling policy, which assigns priority based on the release time of each job. Another study [15] introduced tardiness bounds for the case of fixed priority global scheduling on identical multiprocessors where successive jobs of the same task are allowed to execute concurrently; both preemptive and non-preemptive variations were considered under the sporadic task model. The work by Ahmed and Anderson [1] recently introduced a pseudo-polynomial simulation-based strategy for computing exact bounds for pseudo-harmonic periodic task systems (where every period divides the maximum period) with preemptive *Global-EDF-Like* scheduling algorithms, a class of policies that includes both G-EDF and FIFO. Another study [5] tackles the problem of *gang scheduling*, which imposes groups of tasks to be executed in parallel; in particular, authors derive tardiness bounds for G-EDF in this specific setup.

To the best of our knowledge, this paper is the first study that analytically characterizes the tardiness of uniform instances.

3 PROBLEM DEFINITION

As anticipated in the introduction, in this paper we study the tardiness of a subset of instances of a classical scheduling problem of periodical tasks on a multiprocessor (see Fig. 1 for an example),

called *uniform scheduling instances*. We consider set τ of N synchronous tasks $\tau_1, \tau_2, \ldots, \tau_N$, where each task τ_i consists of an infinite sequence of identical periodical jobs $\tau_{i0}, \tau_{i1}, \tau_{i2}, \ldots$. All tasks have the same period length, denoted by $P > 0$ and expressed in time units. We will refer to period j to indicate the time spanning from jP to $(j+1)P$. The job inter-arrival time is the task period: the j^{th} job arrives at the beginning of the j^{th} period and its deadline is the end of the same period. Jobs of different tasks have the same length, denoted by $L > 0$, which specifies the number of time units needed to execute the job on a unit-speed processor. There are *precedence constraints* on the execution order of jobs of the same task: for any i, a job τ_{ij} cannot be executed if the previous job $\tau_{i(j-1)}$ has not yet finished.

Jobs are to be scheduled using the non-preemptive G-EDF policy (i.e., the next job assigned to the first idle processor is the one with the closest deadline) on a symmetric multiprocessor composed of M identical, unit-speed processors. Ties are broken arbitrarily, e.g. by choosing the task with the smallest index. Non-preemptive means that an executing job must finish on the same processor with no interruptions. We assume that the number of processors is less than the number of tasks, $M < N$.

Intuitively, the *tardiness* of a schedule is the maximum, over all periods, processors, and jobs, of the difference between a job completion time and its deadline. If no job ever exceeds its deadline, tardiness is set to zero. Our goal is to study the tardiness of uniform scheduling instances. It can be proven that, if the amount of work to be done in each period is greater than the available execution time in the period then tardiness is not upper-bounded but grows indefinitely. Therefore in the following, we will also assume that $NL \leq MP$.

Definition 3.1 (Uniform Instance). An *Uniform Instance* is a tuple (N, L, M, P) of positive integers such that

$$M < N, L \leq P \text{ and } NL \leq MP,$$

where M is the number of processors, N is the number of tasks, and tasks synchronously release one job of length L every P time units, with deadline the end of the period.

Definition 3.2 (G-EDF Uniform Scheduling Tardiness Problem). Given a uniform instance I, the *G-EDF Uniform Scheduling Tardiness Problem* is to determine the tardiness value deriving from scheduling instance I with the G-EDF scheduling policy.

Given a uniform scheduling instance, it is possible to prove that precedence constraints (between successive jobs of the same task) never cause any idle time, i.e. a processor is idle if only if there are no pending jobs.

Finally, observe that with uniform instances the priority criterion of G-EDF is the same as the one used by First In First Out (FIFO), a simple scheduling policy that sorts jobs based on their arrival time, therefore our results directly applies also to FIFO scheduling policy.

4 NOTATION AND PRELIMINARIES

In this section, we present some preliminary results that will be used in the following sections. To avoid being tedious, notation is introduced when needed and summarized for the reader's convenience in Table 1.

Table 1: Uniform Instance Parameters

N	Number of tasks
L	length of tasks jobs (same of all jobs)
M	Number of processors
P	Length of the period (same for all tasks)
R_{mj}	Tardiness of processor m at the end of period j
R_j	Maximum tardiness at the end of period j
T	Maximum tardiness among all periods
λ	Tardiness caused by $\lceil \frac{N}{M} \rceil$ jobs with no initial tardiness
μ	Slack time after $\lfloor \frac{N}{M} \rfloor$ jobs with no initial tardiness
Λ_{mj}	Lateness of processor M at the end of period j
Δ_j	Maximum difference in lateness at the end of period j

We start by formally defining the concept of tardiness for different entities:

Definition 4.1 (Tardiness). *Job tardiness* is the difference between the job completion time and its deadline, or zero if the difference is negative. The *tardiness R_{mj} of a processor m* in period j is the maximum tardiness of the jobs of period j assigned to the processor, or zero if no job is assigned to it. The *period tardiness R_j* is the maximum tardiness of all processors in period j (i.e., $R_j = \max_m R_{mj}$), and the *scheduling tardiness* T is the maximum tardiness among periods (i.e., $\max_j R_j = \max_j \max_m R_{mj}$).

Since no job is executed before period 0, we set $R_{-1} = 0$. Observe that tardiness is always non-negative.

We restate the goal of this study by saying that we search for an upper bound to the scheduling tardiness for every uniform scheduling instance.

In the rest of this section, we will first show that job precedence constraints never cause idle time in our context, and thus that we can ignore them in our analysis. Then we start investigating uniform scheduling instances.

4.1 Lateness and precedence constraints

In this section we prove that, given a uniform scheduling instance, the precedence constraints never cause any idle time, i.e. if a processor ended the execution of a job and there are waiting to be executed jobs, then the processor starts executing one of them without idle time.

We define lateness as a value related to processors. Intuitively, lateness is computed as tardiness but it is allowed to be negative.

Definition 4.2 (Lateness). The lateness of a job is the difference between the job completion time and its deadline. The lateness Λ_{mj} of a processor m in period j, is the maximum lateness of the jobs of period j assigned to the processor. If no job is assigned to processor m in period j, its lateness is set to $\Lambda_{m(j-1)} - P$.

Since no job is executed before period 0, we set $\Lambda_{m(-1)} = 0$ for all processors $m \in [1..M]$. In general if for a particular period j a processor m finishes the execution of all jobs j assigned to it before the end of the period, the lateness of that processor is negative for that period: $\Lambda_{mj} < 0$. Note that since lateness is defined by the

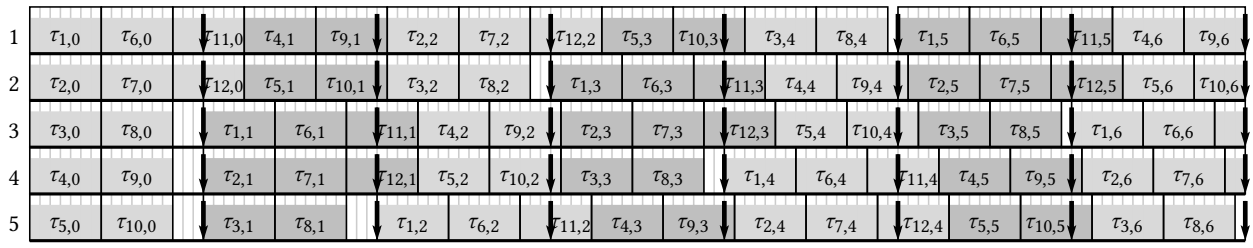


Figure 1: Scheduled example of the instance $(N, L, M, P) = (12, 7, 5, 17)$ in which $(N \mod M) = 2$. Jobs of the same period share the same color. Each row corresponds to a processor, while each column is a time unit. Deadlines are represented by an arrow pointing downwards and they act as a separator of periods. Note that processor 1 increases its tardiness two times and eventually resets it at period 4; after that period the behavior is periodic and the tardiness is always lower than $L = 7$. Moreover, each processor either executes $\lfloor \frac{N}{M} \rfloor = 2$ or $\lceil \frac{N}{M} \rceil = 3$ jobs in each period.

finishing time of a job, and a job has to be executed by a processor in order to finish, by considering the lateness of processors we cover the lateness of all the jobs.

Definition 4.3. (Lateness difference) We define the difference between maximum lateness and minimum lateness of any processor at the end of period j as:

$$\Delta_j = \max_m \Lambda_{mj} - \min_m \Lambda_{mj}. \tag{1}$$

We now show that the maximum difference between the lateness of two jobs of the same period never exceeds the length of a job L. This in turn allows us to conclude that precedence constraints never cause idle time.

Lemma 4.4. *Let $j \geq 0$, if $\Delta_{j-1} \leq L$, then in period j there are no delays caused by precedence constraints, i.e. no processor can be idle if the job queue is not empty.*

Proof. Note that all processors must have assigned at least one job. In fact, if this was not the case, by the pigeonhole principle one processor would execute at least $\lceil \frac{N}{M-1} \rceil \geq 2$ jobs; this in turn would mean that $\Delta_{j-1} \geq 2L$, which is a contradiction. Consider a job of period $j-1$, $\tau_{i(j-1)}$, for some $i \in [1..N]$, executed by processor $m \in [1..M]$. As soon as m is ready to execute jobs of period j, that is at time $t = jP + R_{m(j-1)}$, at least one job will not have to wait for the predecessor to finish executing, namely job τ_{ij}. So the first job executed by each processor at period j is not subject to delays caused by the precedence constraints.

Since $\Delta_{j-1} \leq L$, once the first job of period j is finished by the processor with minimum lateness, that is at time

$$\begin{aligned} & jP + \max\left(0, \min_m \Lambda_{m(j-1)}\right) + L \\ \geq\ & jP + \min_m \Lambda_{m(j-1)} + L \\ \geq\ & jP + \min_m \Lambda_{m(j-1)} + \Delta_{j-1} \\ \overset{(1)}{=}\ & jP + \max_m \Lambda_{m(j-1)}, \end{aligned}$$

all jobs of period $j-1$ will be finished and thus will not cause delays by the precedence constraints. □

Theorem 4.5. *Let $j \geq 0$, in period j the lateness difference is lower or equal than L: $\Delta_j \leq L$.*

Proof. We use induction on all periods.

Base of the induction: In the first period all jobs arrive at time 0. As soon as any processor finishes executing a job and becomes idle, the scheduling policy assigns it a job, if available. This means that each processor gets assigned at least $\lfloor \frac{N}{M} \rfloor \geq 1$ jobs. The remaining $(N \mod M) < M$ jobs, if any, are distributed to some of the processors. This implies that at period 0 the difference between the minimum and the maximum lateness can be 0 or L. Formally $\Delta_0 \leq L$.

Inductive hypothesis: $\Delta_{j-1} \leq L$, $j \geq 1$.

Inductive step: $\Delta_j \leq L$.

By lemma 4.4 and inductive hypothesis, there is no idle time between the execution of two jobs on the same processor at period j. We now prove the Inductive step by contradiction. Suppose that the difference between the minimum and maximum lateness exceeds the length of a job at period j. This implies that after finishing the job that causes minimum lateness, precisely at time

$$jP + \Lambda_{m^* j} + 1 = jP + \min_m \Lambda_{mj} + 1,$$

at least one processor m^* is idle. At the same time the processor that has maximum tardiness at period j has not started yet its last job. So m^* is idle when the job queue is not empty. A contradiction. □

Corollary 4.6. *Precedence constraints never cause idle time in uniform instances.*

Therefore, we will not be concerned about job precedence in the rest of the paper.

4.2 Uniform instances

To study the bound to the schedule tardiness, we introduce the following values related to uniform instances that will be used throughout the paper (see Fig. 2 for an explicating example).

Definition 4.7 (Values λ and μ). For a given uniform instance (N, L, M, P), we define

$$\lambda = \left\lceil \frac{N}{M} \right\rceil L - P,$$

and

$$\mu = P - \left\lfloor \frac{N}{M} \right\rfloor L.$$

Intuitively, when $0 < \lambda \leq L$, then value λ denotes how much executing $\lceil \frac{N}{M} \rceil$ jobs exceeds the length of the first period. When $0 < \mu \leq L$, then μ is the remaining time of the first period after executing $\lfloor \frac{N}{M} \rfloor$ jobs.

Remark 1. If $(N \mod M) \neq 0$, then the sum of λ and μ is equal to L (see also Fig. 2). Indeed:

$$\lambda + \mu = \left\lceil \frac{N}{M} \right\rceil L - P + P - \left\lfloor \frac{N}{M} \right\rfloor L = \left(\left\lceil \frac{N}{M} \right\rceil - \left\lfloor \frac{N}{M} \right\rfloor \right) L = L$$

that is true if and only if $(N \mod M) \neq 0$.

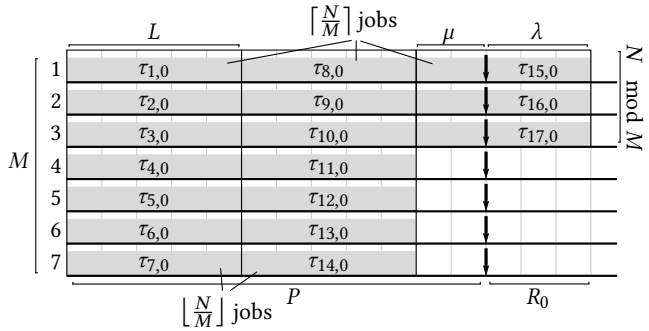


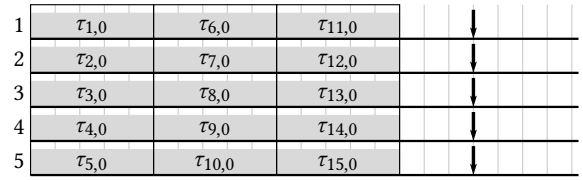
Figure 2: Scheduled example in period 0 of the instance $(N, L, M, P) = (17, 5, 7, 12)$ in which $(N \mod M) = 3$, $\lambda = 15 - 12 = 3$ and $\mu = 12 - 10 = 2$. Note that since $(N \mod M) \neq 0$, λ and μ sum to L, and that each processor executes either $\lfloor \frac{N}{M} \rfloor$ or $\lceil \frac{N}{M} \rceil$ jobs.

Remark 2. Observe that, if $(N \mod M) > 0$, we have that

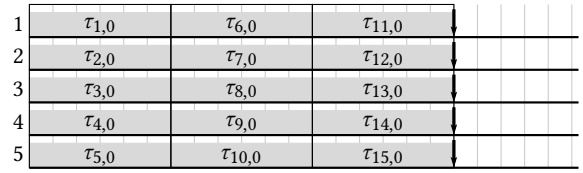
$$\left\lceil \frac{\lambda}{\mu} \right\rceil + 1 = \left\lceil \frac{\lambda + \mu}{\mu} \right\rceil = \left\lceil \frac{L}{\mu} \right\rceil.$$

Easy cases. There are some instances for which it is easy to state that the schedule tardiness is always zero (and will no longer be taken into consideration in the rest of the paper after this section) because it is always possible to schedule all jobs of each period within the period. These instances are characterized by the following parameter values:

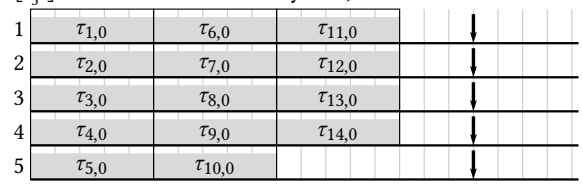
- If $(N \mod M) = 0$ we have $\lfloor \frac{N}{M} \rfloor = \lceil \frac{N}{M} \rceil = \frac{N}{M}$ (see Figure 3a). In each period, each processor executes exactly $\frac{N}{M}$ jobs and terminates $\frac{N}{M} L < P$ time units after the beginning of the period, that is before the end of the period.
- If $\mu = 0$ then we have $P = \lfloor \frac{N}{M} \rfloor L$ (see Figure 3b), implying $(N \mod M) = 0$, and the previous result applies also in this case. Since $P \geq \frac{N}{M} L \geq \lfloor \frac{N}{M} \rfloor L = P$, we have that $\frac{N}{M} L = \lfloor \frac{N}{M} \rfloor L$. The difference with the former case is that here we use all available computation time, i.e., $NL = MP$.
- If $(N \mod M) \neq 0$ and $\mu \geq L$ or if $\lambda \leq 0$ then in each period it is always possible to assign $\lceil \frac{N}{M} \rceil$ jobs to $(N \mod M)$ processors and $\lfloor \frac{N}{M} \rfloor$ to the others, incurring in zero tardiness (see Figure 3c).



(a) Period 0 of instance $N = 15, L = 5, M = 5, P = 18$, in which $(N \mod M) = 0$. Since this is an easy case, no tardiness is created.



(b) Period 0 of instance $N = 15, L = 6, M = 5, P = 18$, in which $\mu = 18 - \lfloor \frac{15}{5} \rfloor 6 = 0$. Since this is an easy case, no tardiness is created.



(c) Period 0 of instance $N = 14, L = 5, M = 5, P = 18$, in which $\lambda = \lceil \frac{14}{5} \rceil 5 - 18 \leq 0$. Since this is an easy case, no tardiness is created.

Figure 3: Graphical representation of easy instances. Only the first period is scheduled, the successive are identical.

Difficult cases. As a consequence of the characterization of easy cases, in the rest of the paper we will always assume that μ, λ and $(N \mod M)$ are all strictly positive, and that μ is smaller than L. These parameter values define difficult instances.

Definition 4.8 (Available processor). A processor is said to be *available* in a period if its tardiness is equal to zero at the beginning of that period.

The following lemma is fundamental in several further proofs. It states that if a period starts with tardiness smaller than the length of a job L, then each processor is assigned either $\lfloor \frac{N}{M} \rfloor$ or $\lceil \frac{N}{M} \rceil$ jobs. Fig. 1 shows an example with $(N, L, M, P) = (12, 7, 5, 17)$, in which $\lambda = 21 - 17 = 4$ and $\mu = 17 - 14 = 3$; processors are assigned at least two jobs ($\lfloor \frac{N}{M} \rfloor = \lfloor \frac{12}{5} \rfloor = 2$) and at most three jobs in each period ($\lceil \frac{N}{M} \rceil = \lceil \frac{12}{5} \rceil = 3$).

Lemma 4.9. *Let $j \geq 0$, if $R_{j-1} < L$, then in period j exactly $(N \mod M)$ processors are assigned $\lceil \frac{N}{M} \rceil$ jobs, while the others only $\lfloor \frac{N}{M} \rfloor$.*

Proof. Available processors (if any) are assigned their first job at the beginning of the period, and will not end their execution before the non-available ones end the execution of the jobs of the previous period, because $R_{j-1} < L$. As long as there are jobs to execute, non-available processors will be assigned their first job of period j with the offset given by their tardiness but always while the others are still executing their first job of period j, again

because $R_{j-1} < L$. And analogously for jobs of period j successively assigned to processors.

In general, no processor finishes executing its k^{th} job of period j before any other processor finishes executing its $(k-1)^{th}$ job of the same period, for $k \in \{1, 2, \ldots, \lceil \frac{N}{M} \rceil\}$. Thus at the end of the period, by a counting argument, $(N \mod M)$ processors are assigned $\lceil \frac{N}{M} \rceil$ jobs of period j, while the remaining $M - (N \mod M)$ processors are assigned $\lfloor \frac{N}{M} \rfloor$ jobs of the same period. □

Definition 4.10 (Selected and non-selected processor). A processor that is assigned $\lceil \frac{N}{M} \rceil$ (resp. $\lfloor \frac{N}{M} \rfloor$) jobs in a given period is said to be *selected* (resp. *non-selected*), and the action of assigning such a number of jobs to the processor is referred to as *selecting* (resp. *non-selecting*) the processor.

Definition 4.11. We refer to the tardiness *resetting* when its value goes back to zero after having been positive for a number consecutive of periods.

Corollary 4.12. *Let $j \geq 0$, if $R_{j-1} < L$, then the tardiness of a selected (resp. non-selected) processor in period j increases (resp. decreases) by λ (resp. μ or resets), and will never be negative.*

Proof. By definition of λ and μ, and because $(N \mod M) > 0$, we have:

(i) if processor m is selected, it increases its tardiness by λ

$$R_{mj} = R_{m(j-1)} + L \left\lceil \frac{N}{M} \right\rceil - P = R_{m(j-1)} + \lambda > 0.$$

(ii) if processor m is not selected, it decreases its tardiness by μ or resets:

$$\begin{aligned} & R_{m(j-1)} + L \left(\left\lfloor \frac{N}{M} \right\rfloor \right) - P \\ = \; & R_{m(j-1)} - \left(P - \left\lfloor \frac{N}{M} \right\rfloor L \right) \\ = \; & R_{m(j-1)} - \mu \end{aligned}$$

and

$$R_{mj} = \max(R_{m(j-1)} - \mu, 0) \geq 0;$$

i.e., since tardiness cannot be negative by definition, if a non-selected processor starts with tardiness lower than μ, it ends its execution before the deadline and causes no tardiness in period j.

□

We conclude this section with a lemma that implies that, if the scheduling tardiness is smaller than L, then processors eventually will reset their tardiness.

Lemma 4.13. *Given an instance (N, L, M, P) let T be the schedule tardiness. If $T < L$, then there exists a finite value $u \in \mathbb{N}$ that corresponds to the number of times a processor increases its tardiness before resetting it.*

Proof. By Corollary 4.12 in each period the tardiness of any processor increases by λ or decreases by μ (possibly resetting) w.r.t. the tardiness of the previous period. By Remark 1 and by hypothesis, a selected processor m must have initial tardiness $\in [0..\mu - 1]$. This is sufficient to observe that the tardiness of processor m at the end of a period j where m is selected is $R_{mj} \in \{(i\lambda \mod \mu) + \lambda : i \in \mathbb{N}\}$. In particular there exists a finite value $ord(\lambda)$ called additive order of $\lambda \in \mathbb{Z}_\mu$ such that $ord(\lambda) \cdot \lambda \equiv 0 \mod \mu$. Since a processor might reset its tardiness or never gain it, it follows that the number of tardiness increases u does not exceed the order: $u \leq ord(\lambda)$, thus u exists and is finite. □

5 TARDINESS CHARACTERIZATION

The main result of our study is to prove that the schedule tardiness for any uniform instance is always bounded by the job length L.

Theorem 5.1. *Given a uniform instance (N, L, M, P), let T be the schedule tardiness. Then we have $T < L$.*

In the next section, we will prove that the bound is tight by showing that there exists an infinite set of instances for which tardiness is exactly $L - 1$.

Moreover, we are able to characterize the processor tardiness for each period of every instance, by providing the exact set of possible period tardiness values. For this purpose, we partition the instance space into an infinite number of disjoint subsets (referred to as *classes* from now on) and show how the period tardiness characterization depends on the class instances belong to. Recall that in this section we are considering only difficult cases, i.e., instances in which the values μ, λ and $(N \mod M)$ are all strictly positive and μ is smaller than L.

Definition 5.2 (Instance space partition). For each $u^* = 1, 2, \ldots$, we define the following class of instances (see also Fig. 4):

$$I_{u^*} = \left\{ (N, L, M, P) : u^* = \min \left\{ u \in \mathbb{N}^+ : \left\lceil \frac{uL}{\mu} \right\rceil \leq \frac{uM}{N \mod M} \right\} \right\}.$$

Theorem 5.3. *Given the instance $(N, L, M, P) \in I_{u^*}$, let T be the schedule tardiness. If $T < L$, then u^* is the exact number of times the tardiness of a processor increases before resetting.*

By looking at the activity of a specific processor and its tardiness values, we observe a periodic behavior that depends on value u^* and can be fully characterized, provided that u^* is known. Finally, we show that u^* is bounded and it is efficiently computable in $O(\min(M, L))$ time.

Corollary 5.4. *If $T < L$, then the tardiness values (R_{mj}) that appear for processor $m \in \{1..M\}$ at the end of any period $j \geq 0$ are either zero or those in the following (finite) set:*

$$\left\{ i\lambda - k\mu \; : \; i, k \in \mathbb{N}, i \in \left[1..u^*\right], \; k \in \left[\left\lfloor \frac{(i-1)\,\lambda}{\mu} \right\rfloor .. \left\lfloor \frac{i\lambda}{\mu} \right\rfloor \right] \right\}.$$

Note that value i depends on the class the instance belongs to, while k spans in an interval that makes the tardiness never exceed value L. Observe that elements of this set can be enumerated in time linear with u^* and therefore the exact tardiness can be computed as the maximum processor tardiness efficiently.

Due to space limitations, in this paper we do not present the general proof of Theorem 5.1. In section 6 we will present our results for a large cardinality class of uniform instances and, while in Section 7 will present the proof of Theorem 5.3 about period tardiness characterization for the other instances. Missing results will be included in the extended version of this paper.

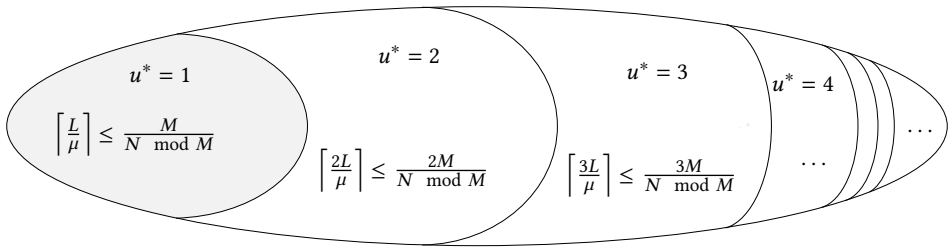


Figure 4: Graphical representation of the classification based on u^*; each instance (N, L, M, P) is assigned to the leftmost class of which it satisfies the related inequality. For the highlighted class we provide full proof of tardiness result.

We conclude this section with the following result that extends our study to instances in which job lengths are not uniform.

Corollary 5.5. *Let $(N, \{L_i\}, M, P)$ be an instance in which each task τ_i produces job with job length L_i. Let $L_{max} = \max L_i$ be the maximum job length. If $NL_{max} \leq MP$, then it is possible to schedule jobs so that tardiness is bounded by the maximum job length L_{max}.*

The proof follows easily by elongating all jobs to L_{max} and using G-EDF over these elongated jobs. This result can be interesting especially for instances where the variance of job lengths is small. Indeed, elongating jobs will not incur in a great increase of utilization. Hopefully the new utilization will not exceed the system computation capacity, and the result might be applied also to instances with large utilization.

Intuition might suggest that directly scheduling such instances (i.e., without elongation) with G-EDF would not produce larger tardiness, as the total amount of work would be lower. However, scheduling anomalies might arise after the first periods: jobs might be assigned to different available processors than in the uniform scheduling, and this could cause a delay in the execution of other jobs later on. Intuition might be right, but the proof is still ongoing research.

6 THE LAMBDA CLASS

In this section, we present the proofs of our results for the class with $u^* = 1$, which is called Lambda class as, we will see, its tardiness is always equal to λ for each period j. See Fig. 5 for an example.

Definition 6.1 (Lambda Class). The Lambda class is the class resulting by setting $u^* = 1$ in definition 5.2, i.e., such that

$$\left\lceil \frac{L}{\mu} \right\rceil \leq \frac{M}{N \mod M}.$$

We will prove that the Lambda class contains instances in which once the tardiness of a processor becomes greater than zero, then it will decrease in successive periods until it is reset again. Only after that, tardiness will increase again. In particular, the maximum processor tardiness is equal to λ and it is decreased exactly by μ time units in successive periods.

Intuitively, the proof is based on the idea that a selected processor m starting with tardiness equal to λ is allowed to repeatedly decrease its tardiness in successive periods until it is reset because it will not be selected for a "sufficient" number of consecutive periods, where the sufficient number of periods is $\left\lceil \frac{\lambda}{\mu} \right\rceil$. This is possible because we are able to prove that there are "enough" other processors that are selected while m is resetting its tardiness, where enough means at least $(N \mod M)$ distinct processors per period. Recall that we assume that value λ is positive.

Lemma 6.2. *In period j, with $j \in \left[0.. \left\lfloor \frac{M}{N \mod M} \right\rfloor - 1\right]$, we have at least $(N \mod M)$ available processors, and $R_{j-1} = \lambda$ for $j > 0$ and $R_{-1} = 0$.*

Proof. At the beginning of period $j = 0$, M processors are available because by definition $R_{-1} = 0$. By lemma 4.9, exactly $(N \mod M)$ processors are selected to execute $\left\lceil \frac{N}{M} \right\rceil$ jobs and cause a tardiness of $\lambda < L$. This means that in the following period if enough processors are available, the same Lemma can be applied. In general, as long as $M - j(N \mod M) > (N \mod M)$, Lemma 4.9 applies and the tardiness at the end of period j is set to λ. This is the case for $j \in [0.. \left\lfloor \frac{M}{N \mod M} \right\rfloor - 1]$. Note that the number of not-yet-selected processors at the beginning of period $\left\lfloor \frac{M}{N \mod M} \right\rfloor$ are exactly $M - \left\lfloor \frac{M}{N \mod M} \right\rfloor \cdot (N \mod M) = (M \mod (N \mod M)) \geq 0$ which are less than $(N \mod M)$.

□

Lemma 6.3. *Given an instance of the Lambda class such that $\lambda > 0$, at the beginning of each period $j = 0, 1, \ldots$, there are at least $(N \mod M)$ available processors and the tardiness of period j is exactly λ.*

Proof. The proof is by strong induction.
Base of the induction: by Lemma 6.2 the thesis is true for each period j, with $1 \leq j \leq \left\lfloor \frac{M}{N \mod M} \right\rfloor - 1$.
Induction hypothesis: we have at least $(N \mod M)$ available processors and $R_{t-1} = \lambda$ for every period t such that $t \in [1..j]$ and $j \geq \left\lfloor \frac{M}{N \mod M} \right\rfloor - 1$.
Induction step: We prove the thesis for period $j + 1$.

By inductive hypothesis and Lemma 4.9, we have that in period j (i) there are $(N \mod M)$ available processors that are selected and incur in tardiness equal to λ; (ii) the remaining $M - (N \mod M)$ processors start the period with tardiness $\leq \lambda$, execute $\left\lfloor \frac{N}{M} \right\rfloor$ jobs and possibly reduce their tardiness by μ. Therefore, $R_j = \lambda$.

As for available processors in period $j+1$. If there are at least $(N \mod M)$ we are done. Otherwise, assume by contradiction there are less than $(N \mod M)$ non-selected processors in period j that do not end their execution before the end of the period. This means that there are at least $s_1 = M - (N \mod M) + 1$ processors that

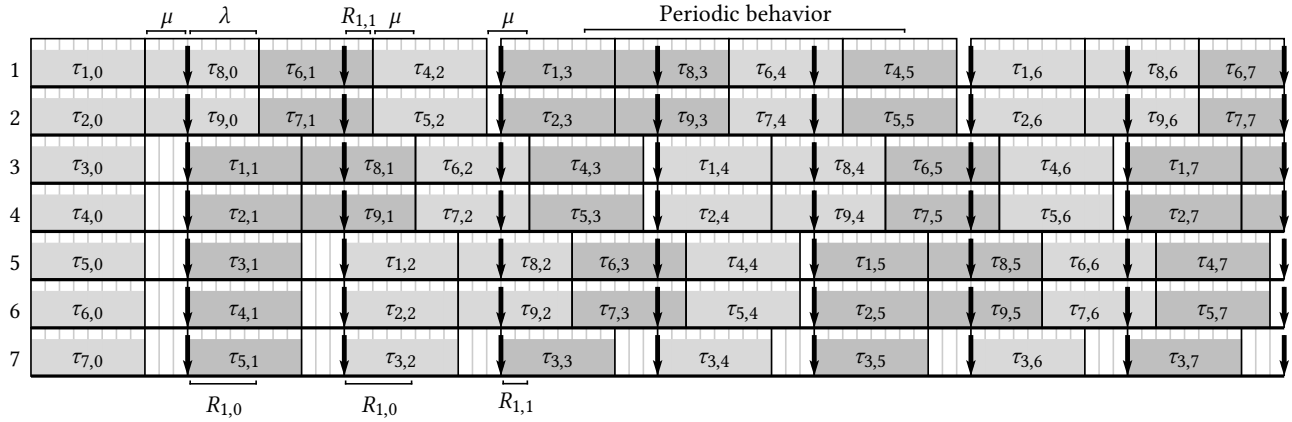


Figure 5: Example of an instance in the Lambda Class. We have: $N = 9, L = 8, M = 7, P = 11$ and $u^* = 1$; since $\mu = 3$ and $(N \mod M) = 2$ we can verify that $\lceil \frac{7}{3} \rceil \leq \frac{3}{1}$. Observe that: at the end of period 0, processors 1 and 2 have tardiness equal to λ; in period 1 processors 3 and 4 have tardiness equal to λ; in period 2 processors 5 and 6 have tardiness equal to λ. Starting from period 3, the job assignments to the processors are cyclic, thus tardiness is easily determined. In general, we can see that the lateness of processor 1 increases once by λ and decreases twice by μ.

have been selected at least once after period $j - \left\lceil \frac{\lambda}{\mu} \right\rceil$ and before period $j + 1$, or they would have reset their tardiness by the end of period j.

By inductive hypothesis and Lemma 4.9, in each period starting from $j - \left\lceil \frac{\lambda}{\mu} \right\rceil + 1$ up to j, there have been exactly $(N \mod M)$ available processors selected per period, for a total of $s_2 = (N \mod M) \cdot \left\lceil \frac{\lambda}{\mu} \right\rceil$ processors. Indeed, s_2 is the exact number because the index of the first considered period is strictly positive:

$$
\begin{aligned}
& j - \left\lceil \frac{\lambda}{\mu} \right\rceil + 1 \quad \text{using Remark 2} \\
&= j - \left\lceil \frac{L}{\mu} \right\rceil + 2 \quad \text{using Lambda class definition (6.1)} \\
&\geq j - \frac{M}{N \mod M} + 2 \\
&\geq \left\lfloor \frac{M}{N \mod M} \right\rfloor - 1 - \frac{M}{N \mod M} + 2 > 0,
\end{aligned}
$$

We now have a contradiction because $s_2 < s_1$:

$$
\begin{aligned}
& s_1 = M - (N \mod M) + 1 > (N \mod M) \cdot \left\lceil \frac{\lambda}{\mu} \right\rceil = s_2 \\
\Leftrightarrow\ & M + 1 > (N \mod M) \cdot \left(\left\lceil \frac{\lambda}{\mu} \right\rceil + 1 \right) = (N \mod M) \cdot \left\lceil \frac{L}{\mu} \right\rceil \\
\Leftrightarrow\ & M \geq (N \mod M) \cdot \left\lceil \frac{L}{\mu} \right\rceil \\
\Leftrightarrow\ & \frac{M}{N \mod M} \geq \left\lceil \frac{L}{\mu} \right\rceil,
\end{aligned}
$$

and the last is true by the definition of the Lambda class. □

Theorem 6.4. *Every instance of the Lambda Class has tardiness equal to $\lambda < L$.*

Corollary 6.5. *Theorem 5.1 provides a tight tardiness bound.*

We show that there is a non-empty set of instances in the Lambda class that has tardiness exactly equal to $L - 1$; e.g. the instances (N, L, M, P) such that $\mu = 1$ are part of the Lambda class. Indeed we have

$$
\begin{aligned}
& LN \leq MP \\
\Leftrightarrow\ & LN \leq M(P + 1 - \mu) \\
\Leftrightarrow\ & LN \leq M \left(P + 1 - P + \left\lfloor \frac{N}{M} \right\rfloor L \right) \\
\Leftrightarrow\ & L \left(N - M \left\lfloor \frac{N}{M} \right\rfloor \right) \leq M \\
\Leftrightarrow\ & L(N \mod M) \leq M \\
\Leftrightarrow\ & \frac{L}{\mu} \leq \frac{M}{N \mod M} \\
\Leftrightarrow\ & \left\lceil \frac{L}{\mu} \right\rceil \leq \frac{M}{N \mod M},
\end{aligned}
$$

and $\lambda = L - \mu = L - 1$. Notice that there are at least the instances such as $(N, L, M, P) = (k + 1, k - 1, k, k)$, for any $k > 0$, for which it holds that $\mu = k - \left\lfloor \frac{k+1}{k} \right\rfloor (k - 1) = 1$. By Theorem 6.4 their tardiness is exactly $\lambda = L - 1$.

Corollary 6.6. *Given an instance of the Lambda class with $\lambda > 0$, the tardiness of processor m at the end of each period falls into the following set of values:*

$$
\left\{ \lambda - k\mu : k = 0, \ldots, \left\lfloor \frac{\lambda}{\mu} \right\rfloor \right\} \bigcup \{0\} \quad \forall j \geq 0, m \in [1..M].
$$

Note that corollary 6.6 is a particular case of corollary 5.4 with $u^* = 1$.

7 GENERAL INSTANCES CLASS

In this section, we focus on instances in classes with $u^* > 1$. The general case is more complicated than the Lambda class, as processor

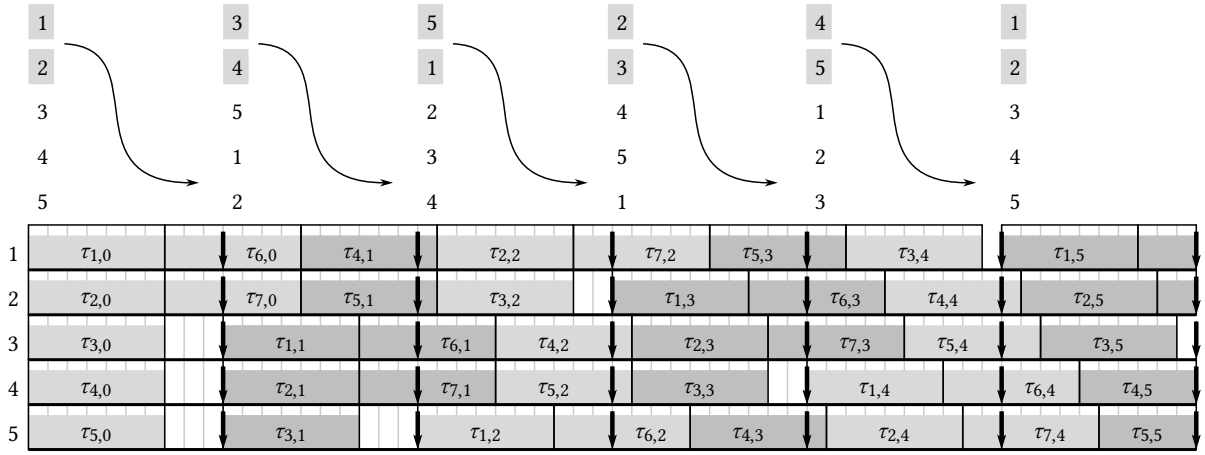


Figure 6: Evolution of the ladder for the instance $(N, L, M, P) = (7, 7, 5, 10) \in I_2$. At the top of the ladder, we can always find the processors with lower initial tardiness. In each period $(N \mod M) = 2$ processors are selected and moved to the bottom of the ladder, while the others are moved to the top by two positions. Note processor 1 increases its tardiness $u^* = 2$ times before resetting it.

tardiness increases more than once before being reset (differently from what happens in the Lambda class).

To prove our results, we build a ladder that periodically classifies processors in function of their initial tardiness. Small tardiness processors score higher on the ladder and ties are broken arbitrarily. By lemma 4.9, if $T < L$, only the first $(N \mod M)$ processors in the ladder increase their tardiness by λ and therefore, in the next period, they are moved at the bottom of the ladder. Indeed, as $T < L$ and $L = \lambda + \mu$, non-selected processors decrease their tardiness to a value strictly smaller than $L - \mu$, while selected processors have tardiness at least $\lambda \geq L - \mu$. All non-selected processors are moved up the ladder by $(N \mod M)$ positions. The described behavior can be observed in Figure 6.

As anticipated, due to lack of space, we are not going to present the proof of Theorem 5.1. However, to give an idea of how the proof works, we briefly describe the five points that are to be proved via induction. Given a generic period $j \geq \left\lfloor \frac{M}{N \mod M} \right\rfloor$ we must show that:

(1) tardiness at the end of the period is smaller than L;
(2) at the end of the period, the tardiness of each processor is given by positive multiples of λ minus positive multiples of μ;
(3) the number of processors that begin period $j + 1$ with tardiness equal to $(i\lambda \mod \mu)$, for any $i \in \mathbb{N}$, is at least $(N \mod M)$;
(4) each selected processor that begins period $j+1$ with tardiness equal to $(i\lambda \mod \mu)$, for some $i \in \mathbb{N}$, is in position $(iM \mod (N \mod M))$ of the ladder;
(5) consider three disjoint groups of processors:
   - selected processors that start period $j - \left\lfloor \frac{M}{N \mod M} \right\rfloor + 1$ with initial tardiness $(i\lambda \mod \mu) \geq (-\lambda \mod \mu)$, for some $i \in \mathbb{N}$;
   - selected processors that start period $j - \left\lfloor \frac{M}{N \mod M} \right\rfloor + 2$ with initial tardiness $(k\lambda \mod \mu) < (-\lambda \mod \mu)$, for some $k \in \mathbb{N}$;
   - non-selected processors that start period $j + 1$ with initial tardiness less than μ.

   The number of such processors is at least $(N \mod M)$.

As an example, consider period $j = 4$ in Figure 6. Note that $4 \geq \left\lfloor \frac{M}{N \mod M} \right\rfloor = \left\lfloor \frac{5}{2} \right\rfloor = 2$; also $\lambda = 4$ and $\mu = 3$. We can see that:

(1) tardiness at the end of the period is $R_4 = 5 < 7 = L$;
(2) $R_{1,4} = 0$, $R_{2,4} = 1 = 4 - 3$, $R_{3,4} = 2 = 2 \cdot 4 - 2 \cdot 3$, $R_{4,4} = 4$ and $R_{5,4} = 5 = 2 \cdot 4 - 3$;
(3) at the beginning of period 5 processors 1, 2 and 3 have tardiness 0, $1 = (4 \mod 3)$ and $2 = (2 \cdot 4 \mod 3)$ respectively; since $(N \mod M) = (7 \mod 5) = 2$, the condition is met;
(4) at the beginning of period 5 processor 1 has initial tardiness $R_{1,4} = 0$ and position 0, while processor 2 has initial tardiness $R_{2,4} = 1 = (1 \cdot 4 \mod 3)$ and position $1 = (1 \cdot 5 \mod 2)$.
(5) (i) in period $4 - 2 + 1 = 3$, there are no selected processors with initial tardiness smaller than $2 = (-4 \mod 3)$; (ii) in period $4 - 2 + 2 = 4$, processors 4 and 5 are selected and their initial tardiness is lower than $2 = (-4 \mod 3)$; (iii) in period 5, only processor 3 starts with tardiness $2 < 3$ and is not selected; The total number of considered processors is 3, which is greater than $(7 \mod 5) = 2$.

We conclude this section with the proof of Theorem 5.3, which states that the tardiness of each processor increases exactly u^* times before resetting under the assumption $T < L$, given by Theorem 5.1.

Proof of Theorem 5.3. Let $u \in \mathbb{N}^+$ be the number of times processor m increases its tardiness before resetting it. By lemma 4.13 u is guaranteed to exist. W.l.o.g., assume m starts at the very first position of the ladder with tardiness zero. By hypothesis we have $T < L$, thus by Corollary 4.12, processor m will be subject to

interleaved tardiness increases and decreases for exactly $u + \left\lceil \frac{u\lambda}{\mu} \right\rceil = \left\lceil \frac{uL}{\mu} \right\rceil$ periods in order to reset the tardiness cumulated in the u increases: u is the number of periods in which m is selected, and $\left\lceil \frac{u\lambda}{\mu} \right\rceil$ is the cumulative number of periods needed to reduce the tardiness to a value in the range $[0..\mu - 1]$ and then reset it.

By Lemma 4.9, exactly the top $(N \mod M)$ processors of the ladder are selected in each period, so every time m is selected, it is moved to the bottom of the ladder, and it takes $\left\lfloor \frac{M}{N \mod M} \right\rfloor - 1$ periods to climb back to the top $(N \mod M)$ positions. During these periods other processors will be selected and be placed at the bottom of the ladder (namely the top $(N \mod M)$ ranks of each period). To account for the u tardiness increases, m needs $u + \left(\left\lfloor \frac{uM}{N \mod M} \right\rfloor - u \right) = \left\lfloor \frac{uM}{N \mod M} \right\rfloor$ periods to gain the top $(N \mod M)$ positions of the ladder $u + 1$ times By this time m must have reset its tardiness (or it would increase it $u+1$ before resetting). Therefore, it must be

$$u + \left\lceil \frac{u\lambda}{\mu} \right\rceil \leq u + \left(\left\lfloor \frac{uM}{N \mod M} \right\rfloor - u \right)$$

and

$$\left\lceil \frac{uL}{\mu} \right\rceil \leq \left\lfloor \frac{uM}{N \mod M} \right\rfloor \leq \frac{uM}{N \mod M}, \tag{2}$$

implying $u \geq u^*$. Moreover, by the same time, m reset its tardiness only once, therefore, u must be the smallest value for which (2) holds. Thus it must be $u = u^*$. □

Now we show that for every instance we can effectively find its class, as the value u^* is always finite, in time that linearly depends on the minimum among the number of processor and the job length.

Lemma 7.1. *Given instance (N, L, M, P), value u^* is bounded and can be computed in $O(\min(M, L))$.*

Proof. Let $\chi = \mu / \gcd(L, \mu)$. Since

$$L(N \mod M) = L\left(N - \left\lfloor \frac{N}{M} \right\rfloor M\right) \leq PM - \left\lfloor \frac{N}{M} \right\rfloor ML = M\mu,$$

we have that χ satisfies the condition that defines u^*. In fact:

$$\begin{aligned} &\left\lceil \frac{\chi L}{\mu} \right\rceil \\ &= \left\lceil \frac{\mu}{\gcd(L, \mu)} \frac{L}{\mu} \right\rceil \\ &= \frac{\mu}{\gcd(L, \mu)} \frac{L}{\mu} \\ &\leq \frac{\mu}{\gcd(L, \mu)} \frac{M}{N \mod M} \\ &= \frac{\chi M}{N \mod M}, \end{aligned}$$

thus χ is an upper bound of u^*. Similarly one can show that the value $\frac{N \mod M}{\gcd(N,M)}$ is another upper bound of u^*. Since $(N \mod M) < M$ and $\mu < L$ (if $u^* > 1$), we can find the value of u^* by testing each value between 1 and the minimum of the mentioned upper bounds. □

Note that $O(\min(M, L))$ is pseudo-polynomial in the length of the input. Nevertheless, to determine the classes of instances arising in practical applications is still computationally tractable because $O(\min(M, L)) \in O(M)$ and the number of processors is relatively small in the large majority of interesting applications.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we have analytically studied the tardiness of a subset of instances of a classical soft real-time scheduling problem, namely a harmonic task system with G-EDF scheduling policy where periods and job lengths are uniform, and tasks are synchronous and periodic. We showed that the job length is a tight bound to tardiness and that the exact tardiness value of any uniform instance can be computed in pseudo-polynomial time, linear with the number of processors. This is a result that has interesting implications from both a theoretical and a practical point of view.

As for future work, we will address the problem of finding a closed form to compute u^* in time polynomial with the length of the input, and we plan to continue addressing the study of the tardiness bound from below. We will consider more general instances then uniform ones by dropping constraints one by one. When approaching the problem by successive generalization, we will try to understand why and when the former proofs can not be generalized to the new case under consideration. This should give a deep understanding of the new generalization and might also give rise to hints for its analytic study. For example, when removing jobs lengths uniformity we will still have a clear graphical representation of scheduled instances that is extremely helpful for intuitions and result presentation. On the other hand, we will not be able to drop the job precedence constraints anymore. Further ahead, removing period length uniformity will probably degrade the clarity and usefulness of the graphical representation.

ACKNOWLEDGMENTS

Authors would like to thank Benjamin Rouxel for insightful discussions on the applications of this theoretical study, and the anonymous referee for her/his comment about the pseudo-polynomiality of the computation of u^*.

REFERENCES

[1] Shareef Ahmed and James H Anderson. 2021. Tight Tardiness Bounds for Pseudo-Harmonic Tasks Under Global-EDF-Like Schedulers. In *Proceedings of the 33rd Euromicro Conference on Real-Time Systems*.
[2] Giovanni Buzzega. 2022. *Upper bound alla tardiness di scheduler globali in ambito multiprocessore identico con task omogenei.* Master's thesis. Università degli studi di Modena e Reggio Emilia.
[3] Chien-Ying Chen, Sibin Mohan, Rodolfo Pellizzoni, Rakesh B Bobba, and Negar Kiyavash. 2019. A Novel Side-Channel in Real-Time Schedulers. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 90–102.
[4] UmaMaheswari C Devi and James H Anderson. 2008. Tardiness bounds under global EDF scheduling on a multiprocessor. *Real-Time Systems* 38, 2 (2008), 133–189.
[5] Zheng Dong, Kecheng Yang, Nathan Fisher, and Cong Liu. 2021. Tardiness Bounds for Sporadic Gang Tasks Under Preemptive Global EDF Scheduling. *IEEE Transactions on Parallel and Distributed Systems* 32, 12 (2021), 2867–2879.
[6] Jeremy Erickson, UmaMaheswari Devi, and Sanjoy Baruah. 2010. Improved tardiness bounds for global EDF. In *2010 22nd Euromicro Conference on Real-Time Systems*. IEEE, 14–23.
[7] Jeremy Erickson, Nan Guan, and Sanjoy Baruah. 2010. Tardiness bounds for global EDF with deadlines different from periods. In *International Conference On Principles Of Distributed Systems*. Springer, 286–301.
[8] Jeremy P Erickson. 2014. *Managing tardiness bounds and overload in soft real-time systems.* Ph. D. Dissertation. The University of North Carolina at Chapel Hill.
[9] Jeremy P Erickson and James H Anderson. 2022. Soft real-time scheduling. In *Handbook of Real-Time Computing*. Springer, 233–267.

[10] Mauro Leoncini, Manuela Montangero, and Paolo Valente. 2017. A branch-and-bound algorithm to compute a tighter bound to tardiness for preemptive global EDF scheduler. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. 128–137.

[11] Mauro Leoncini, Manuela Montangero, and Paolo Valente. 2019. A parallel branch-and-bound algorithm to compute a tighter tardiness bound for preemptive global EDF. *Real-Time Systems* 55, 2 (2019), 349–386.

[12] Hennadiy Leontyev and James H Anderson. 2007. Tardiness bounds for FIFO scheduling on multiprocessors. In *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*. IEEE, 71–71.

[13] Hennadiy Leontyev and James H Anderson. 2010. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems* 44, 1 (2010), 26–71.

[14] Gianluca Nocetti. 2021. *Tardiness bound per l'algoritmo G-EDF con job sincroni e lunghezza uniforme.* Bachelor's thesis. Università degli studi di Modena e Reggio Emilia.

[15] Kecheng Yang Sergey Voronov, James H. Anderson. 2018. Tardiness Bounds for Fixed-Priority Global Scheduling without Intra-Task Precedence Constraints. *RTNS '18: Proceedings of the 26th International Conference on Real-Time Networks and Systems* (2018), 8–18.

[16] Paolo Valente. 2016. Using a lag-balance property to tighten tardiness bounds for global EDF. *Real-Time Systems* 52, 4 (2016), 486–561.