Real-Time Systems (2024) 60:537–569
https://doi.org/10.1007/s11241-024-09432-6

Characterizing global work-conserving scheduling tardiness with uniform instances on multiprocessors

Giovanni Buzzega[1] · **Manuela Montangero**[2]

Accepted: 25 September 2024 / Published online: 10 December 2024

Abstract

Soft real-time multiprocessor systems need scheduling policies introducing small overheads and for which it is possible to give guarantees on tardiness (i.e., the maximum delay that might arise with respect to job deadlines) in order to assess their feasibility in specific applications. For these reasons, lightweight policies such as Global Earliest Deadline First, and First-in First-out are preferred. Much effort has been spent in literature to provide efficiently computable tardiness bounds for periodic task systems scheduled on multiprocessors, but still, no exact bounds are known and results are given for specific classes of instances. In this paper, we use a work-conserving policy to schedule uniform instances, namely synchronous and periodic task systems in which tasks have the same period length and the same job length. We analytically derive a tight bound on the maximum tardiness and we give the exact length of the schedule hyper-period, 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.

Keywords Work-conserving scheduling · GEL scheduling · Soft real-time · Tardiness analysis · Hyper-period

Giovanni Buzzega and Manuela Montangero have contributed equally to this work.

✉ Giovanni Buzzega
giovanni.buzzega@phd.unipi.it

✉ Manuela Montangero
manuela.montangero@unimore.it

[1] Department of Computer Science, University of Pisa, Largo B. Pontecorvo, 3, 56127 Pisa, Italy

[2] Department of Physics, Computer Science and Maths, University of Modena and Reggio Emilia, Via Campi 213/B, 41125 Modena, Italy

1 Introduction

Soft Real-Time (SRT) systems are used to model real-world systems that can tolerate, to some extent, deadline misses for their jobs. Such models are useful for a number of applications, including multimedia streaming and industrial automation (Erickson 2014). In SRT systems where responsive execution and predictability are prioritized, one approach to ensure correctness is to consider the *tardiness* of the used scheduling policy, i.e., the maximum amount of time by which a job deadline is missed.

As it is not trivial to derive exact tardiness values, upper bounds are estimated to assert the feasibility of scheduling policy for a task set and specific system requirements a priori: only if the bound is admissible for the application, the scheduling policy might be taken into consideration. Therefore, tightness of tardiness bounds is key to avoiding misclassification of feasible applications: if the upper bound is loose, the scheduling policy might be discarded (in favor of less performing ones) even if it would be able to guarantee the desired performances. Moreover, the scheduling policy chosen for SRT applications should introduce minimal overhead itself, making lightweight priority schemes such as First-in First-out (FIFO) and Global Earliest Deadline First (G-EDF) attractive choices. Global policies allow the scheduling of any job to any processor and are usually preferred when dealing with soft real-time systems, as they allow bounded tardiness if the available processing capacity is not exceeded. This desirable feature is lacking in partitioned scheduling, as statically assigning tasks to processors may lead to global under-utilization of resources and processor overloading, causing unbounded tardiness (Leontyev and Anderson 2009). Therefore, in this paper, we will focus on global scheduling policies.

Different methods have been developed to compute general tardiness bounds for FIFO, G-EDF, and similar policies (Leontyev and Anderson 2007; Devi and Anderson 2008; Valente 2016; Ahmed and Anderson 2021). However, the resulting bounds are often either too complex or demanding to implement and compute in a reasonable time, or too loose to be useful in practice. Therefore, the problem of defining tight tardiness upper bounds is still open.

In this paper, we approach the problem of bounding tardiness for the class of work-conserving non-preemptive global schedulers from below: we restrict our study to a particular class of task sets, and we provide an exact analytical tardiness description of all the considered instances. In addition to this nontrivial result, the provided characterization gives a lower bound to the tardiness on the class of general task systems (i.e., it is not possible to achieve better tardiness, in general), closing the gap from below with respect to the previously given tardiness upper bounds.

In a work-conserving scheduler, processors can not be idle if there are pending jobs (i.e., released and not subject to precedence constraints), and non-preemption does not allow the execution of a job to be suspended once started on a processor. The class of schedulers that we analyze includes widely studied policies such as FIFO, G-EDF, and G-EDF-Like (GEL). These policies differ in the way

they define the order in which released jobs are to be scheduled to processors: they assign a priority to each released job (possibly changing it over time) and schedule the highest priority job first. The FIFO policy gives priority to the jobs that have been released earlier, and the G-EDF policy to the job having the closest deadline. The GEL policies assign priority based on *priority points* that are calculated by means of the task priority and the job release time. Note that the work-conserving and non-preemption features cause the priority of any pending job to be higher than that of any future job, a condition that allows bounded tardiness (Leontyev and Anderson 2009).

As for the set of instances under study in this paper, we consider periodic task systems in which periods and jobs have the same lengths, and synchronous tasks on identical unit-speed processors. We call these task sets *uniform instances*, to reflect their regularity. This places our work in a particular subset of Harmonic Task Systems, i.e. the class of periodic task systems where every period is a multiple of each smaller period (Ahmed and Anderson 2021), which commonly arise in different application fields such as avionics, robotics, control applications and more (Anssi et al. 2013; Fu et al. 2010; Li et al. 2003; Shih et al. 2003).

Even if the set of uniform instances are very specific among harmonic task systems, and they might not naturally arise in the previously mentioned practical applications, we observe that they might find real-world applications whenever security issues are to be taken into consideration. Indeed, uniform instances may be deployed to make tasks indistinguishable for a malicious external observer. Thus, the observer would not be able to easily gather information about the arrival of specific jobs that might be targets of attacks (Chen et al. 2019), because they are all released simultaneously and all have the same execution time. Moreover, imposing the same length on all jobs may help maximize utilization and thus prevent malicious code execution on idle processors, as well as prevent the injection of code to the jobs, as this would be easily detectable.

Contributions. Here we extend our previous work (Buzzega et al. 2023), in which we introduced the problem and proved our results only for a subclass of uniform instances under the G-EDF scheduling policy.

In this paper, we simplify the approach as we tackle the general problem without splitting instances into classes. In this way, we provide the missing proofs and we generalize the results to any global work-conserving non-preemptive scheduler, and we give a complete analytical description of the tardiness for uniform instances. Finally, we provide the hyper-parameter length and the exact maximum tardiness value of any instance, both computable in time linear in the minimum number of processors and the job length. Our results may serve as the basis for future research on exact tardiness bounds for more complex synchronous task sets.

Roadmap. The rest of the paper is organized as follows: we review previous related works from the literature (Sect. 2) and we formally introduce the problem (Sect. 3). After that we introduce notation and preliminaries (Sect. 4) to prove our main result on tardiness in Sect. 5; then, we introduce more notation needed to prove the remaining lemmas in Sect. 6. Finally, we prove our results related to the hyperperiod length in Sect. 7 and we conclude (Sect. 8).

2 Related work

In the literature of soft real-time scheduling, significant research has been conducted on establishing tardiness bounds, and a large part of the studies on global schedulers has focused on G-EDF and GEL policies. Initially, both preemptive and non-preemptive G-EDF policies were proven to be SRT-optimal for identical multiprocessor platforms by Devi and Anderson (2008), i.e. they ensure bounded tardiness for task sets that do not over-utilize available computing time. Subsequent improvements to tardiness bounds for preemptive G-EDF were introduced by Erickson et al. (2010a) using compliant vector analysis, with extensions made for arbitrary sporadic task systems in a later work by Erickson et al. (2010b). Leontyev and Anderson (2009) further demonstrated the SRT-optimality of a broad class of global algorithms that can model dynamically changing priorities of tasks.

More recently, Valente (2016) proposed the *harmonic bound* as an upper limit on tardiness for G-EDF, which was shown to be up to 30% tighter than previous bounds. However, the high computational complexity of deriving this bound limited its practical utility, leading to the development of a Branch-and-Bound algorithm by Leoncini et al. (2017) and its subsequent parallelization in a later work (Leoncini et al. 2019).

Several studies have also focused on specific instances of the tardiness-bound problem to refine their accuracy. For example, Leontyev and Anderson (2007) introduced upper bounds for the FIFO scheduling policy, while Voronov et al. (2018) derived tardiness bounds for fixed-priority global scheduling on identical multiprocessors with concurrent execution of successive jobs within the same task. Dong et al. (2021) addressed the challenge of gang scheduling, where task groups are executed in parallel, by establishing tardiness bounds for G-EDF in such scenarios. Ahmed and Anderson (2021) recently presented a pseudo-polynomial simulation-based approach for exact bound computation in pseudo-harmonic periodic task systems under preemptive GEL schedulers.

Going in the opposite direction, Erickson et al. (2014) and Goh and Anderson (2023) adopt a linear-programming technique to select the parameters of their GEL scheduling policies; the objective is to minimize response-time bounds under compliant vector analysis, that is the time taken to finish the execution of a job from its release.

Other research on tardiness estimates has focused on a more realistic setting where jobs exhibit stochastic execution demand and bounds are derived on expected tardiness (Mills and Anderson 2010, 2011). Some studies take into consideration memory management issues that arise in SRT systems, such as short bursts of high memory bandwidth requests. In multicore architectures, such a phenomenon may cause memory contentions with non-real-time tasks, which in turn could result in missed deadlines. Kato et al. (2011) propose and implement a CPU scheduler based on EDF, and enrich it with a memory reservation scheme that allows real-time tasks to reserve or share memory pages. Yun et al. (2017) propose a memory bandwidth throttling mechanism, which allows the user

to explicitly define segments in which SRT jobs should be given priority. Even though this is an interesting area of research and is useful for practical implementations, it is beyond the scope of this paper. In the following, we propose a purely theoretical and deterministic analysis of tardiness and hide such unexpected delays in the worst-case execution time of each job.

3 The scheduling tardiness problem

In this paper, we study the tardiness of a subset of instances of a classical scheduling problem of periodic tasks on a multiprocessor (see Fig. 1 for an example), called *uniform scheduling instances*, because they are uniform with respect to job and period length. In this section, we present the problem addressed in this paper and we give formal definitions of the main concepts used throughout the paper.

Task model. We consider a 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 $\tau_{i,0}, \tau_{i,1}, \tau_{i,2}, \ldots$. Each job is characterized by: (i) an *arrival (or release) time*, after which the job must be eventually scheduled on some processor; (ii) a *completion time* (or *length*) that is the time needed to execute the job on a unit-speed processor; (iii) an *absolute deadline*, that it the time by when the job execution should be finished. Failing to do so would cause positive *tardiness* for that job. Each task is characterized by a *period*, that defines the job *inter-arrival time*, i.e., the jth job of task i, $\tau_{i,j}$, is released in period j.

Moreover, jobs of the same task are subject to *precedence constraints*: for any task τ_i, job $\tau_{i,j}$ cannot start its execution before job $\tau_{i,j-1}$ has ended.

We model the scheduling problem in the discrete-time domain, meaning that the job arrival time and finish time are non-negative integer values. Within this general context, in this paper, we consider *uniform instances*, having the following characteristics:

- All tasks have the same period length, denoted by $P > 0$ and expressed in time units.
- The job inter-arrival time is the task period: the jth job of any task arrives at the beginning of the jth 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.

As tasks synchronously release their jth job at the same time at the beginning of the period, we will refer to *period* $j \geq 0$ to indicate the time spanning from jP to $(j+1)P - 1$, included.

Scheduling policy. We assume that jobs are to be scheduled using any non-preemptive work-conserving global policy on a symmetric multiprocessor composed of M identical, unit-speed processors. Non-preemptive means that an executing job must finish on the same processor with no interruptions. Work-conserving means that, if there are jobs ready to be executed that are not subject to precedence

constraints, no processor can be idle. We focus on *global* policies, which allow scheduling any job to any idle processor without constraints. Intuitively, this means that in a period where no precedence constraints apply, the earlier a processor is ready to execute jobs of that period, the more jobs it is assigned. Finally, a symmetric multiprocessor is composed of two or more identical processors connected to a single shared memory, in which all processors are treated equally and none are reserved for special purposes.

Throughout the paper, the term *assign* is used to refer to the scheduling of a job on a specific processor to begin executing within a given period.

In this paper, we focus on characterizing the *schedule tardiness* under the previous assumptions and for uniform instances. Intuitively, the tardiness of a schedule is the maximum delay in completing a job (w.r.t. to its deadline) that occurs in the whole (infinite) scheduling of jobs to processors. In other words, tardiness is the maximum of the differences between job completion times and deadlines, taken over all periods, processors, and jobs. If no job ever exceeds its deadline, tardiness is zero.

As in Leontyev and Anderson (2008), in order to avoid unbounded tardiness growth, we assume that the amount of work $N \cdot L$ is not larger than the available execution time $M \cdot P$, i.e., that $NL \leq MP$. To avoid trivial instances, we also assume that the number of processors is not greater than the number of tasks, i.e., that $M \leq N$. Indeed, if we have more processors than tasks, then each task can be executed on a dedicated processor, and tardiness will be zero if the job length is less than P, or diverge due to the precedence constraints whenever $L > P$. The two conditions $NL \leq MP$ and $M \leq N$ together imply that $L \leq P$.

We will see that, thanks to the symmetry of job lengths and the work-conserving nature of the scheduler, we will not need to refer to jobs of specific tasks; instead, since all jobs are eventually executed and can not be preempted, we can shift our focus to processors: for each period and processor, we deal with the tardiness of the last job of that period executed by that processor. Therefore, the term τ will not be explicitly used in the following, and we will work with the number of jobs executed by a processor in a given period.

In the following, we show that any priority policy of a global non-preemptive work-conserving scheduler produces the same maximum *tardiness* on uniform instances and that this bound is given by the length of the job.

3.1 Problem definition

We start by formally defining uniform instances.

Definition 1 (*Uniform Instance*) A *Uniform Instance* is a tuple (N, L, M, P) of positive integers such that

$$M \leq N, \tag{1}$$

$$L \leq P, \tag{2}$$

$$NL \leq MP, \tag{3}$$

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 at the end of the period.

Traditionally in literature, tardiness is defined for the whole scheduling, while in this paper we will decline tardiness also for periods and processors. Thus, we formally define the concept of tardiness for different entities.

Definition 2 (*Tardiness*) We define the following different types of tardiness:

- *Job tardiness* is the difference between the job completion time and its deadline (that is, the end of the release period for uniform instances), or zero if the difference is negative.
- The *tardiness $R_{m,j}$ of a processor m* in period j is the maximum tardiness of the jobs assigned to m in the previous period $j-1$[1]. If no jobs were assigned, the tardiness is considered to be zero. Note that, due to the lack of preemption, each job is assigned to only one processor. We say that the tardiness of processor m *resets* in period j when it drops to zero in period j after being positive for $k \geq 1$ consecutive periods.
- The *tardiness R_j of period j* is the maximum tardiness of all processors in period j, i.e., $R_j = \max_m R_{m,j}$.
- The *scheduling tardiness* T is the maximum tardiness among periods, i.e., $T = \max_j R_j = \max_j \max_m R_{m,j}$.

Observe that tardiness is always non-negative, and since no job is executed before period 0, we set $R_0 = 0$.

Example 1 In Fig. 1, in period 1, processors $1, 2$, and 3 have tardiness $R_1 = R_{1,1} = R_{2,1} = R_{3,1} = 4$. On the other hand processors 4 and 5 have no tardiness: $R_{4,1} = R_{5,1} = 0$.

In the rest of the paper, when we talk about processor or period tardiness values, we refer to it as a feature of the period, as if it is computed a posteriori when all jobs of that period have finished their execution.

Moreover, we shall use extensive comparisons between tardiness values of different periods. In particular when we say that in period j tardiness *increases*, or *decreases*, we refer to changes in the tardiness value of a given period with respect to the one of the previous period.

We are now ready to define the problem addressed in the paper:

[1] Note that the concept of tardiness differs from the one we introduced in the conference version of this paper Buzzega et al. (2023), where $R_{m,j}$ referred to the tardiness of jobs of period j.

Table 1 Notation

N	Number of tasks
L	length of tasks jobs (same of all jobs)
M	Number of processors
P	Length of the period (the same for all tasks)
$R_{m,j}$	Tardiness of processor m at the beginning of period j
R_j	Maximum processor tardiness at the beginning of period j
T	Maximum tardiness among all periods
λ	Tardiness caused by $\left\lceil \frac{N}{M} \right\rceil$ jobs in a period with tardiness equal to zero
μ	Slack time after $\left\lfloor \frac{N}{M} \right\rfloor$ jobs in a period with tardiness equal to zero
A_j	Set of processors with tardiness lower than μ in period j
S_j	Set of processors that are assigned $\left\lceil \frac{N}{M} \right\rceil$ jobs of period j
$\overline{S}_j$	Set of processors that are not assigned $\left\lceil \frac{N}{M} \right\rceil$ jobs of period j

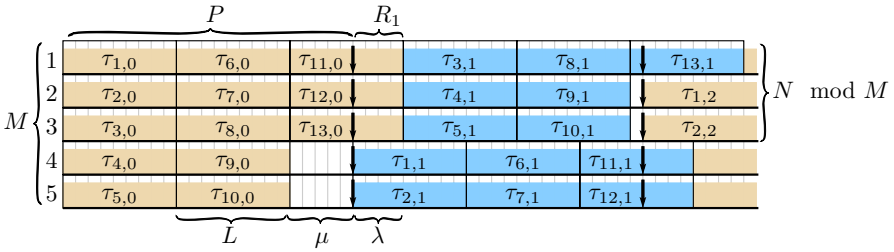


Fig. 1 Scheduled example in the first periods of the instance $(N, L, M, P) = (13, 9, 5, 23)$ in which $(N \mod M) = 3$, $\lambda = 27 - 23 = 4$ and $\mu = 23 - 18 = 5$. Note that since $(N \mod M) \neq 0$, λ and μ sum to L, and that each processor executes either $\left\lfloor \frac{N}{M} \right\rfloor = 2$ or $\left\lceil \frac{N}{M} \right\rceil = 3$ jobs

Definition 3 (*Uniform Scheduling Tardiness Problem*) Given a uniform instance I, the *Uniform Scheduling Tardiness Problem* is to determine the tardiness value T deriving from scheduling instance I with a non-preemptive, work-conserving scheduling policy.

4 Notation and preliminaries

In this section, we first briefly recall some properties of the modulo operator that will be heavily used throughout the paper, and then we continue with the preliminaries.

Modulo operator. For any positive integers a and b, we have that floor and ceiling operators are related depending on the modulo operation:

$$\left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a}{b} \right\rfloor + 1 \Leftarrow (a \mod b) \neq 0, \text{ otherwise } \left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a}{b} \right\rfloor.$$

Moreover, we can rewrite them in long form:

$$\left\lfloor \frac{a}{b} \right\rfloor = \frac{a - (a \mod b)}{b}, \quad \left\lceil \frac{a}{b} \right\rceil = \frac{a + (-a \mod b)}{b}.$$

We emphasize the difference between $(-a \mod b)$ and $-(a \mod b)$: the former is always non-negative and the latter is non-positive; specifically, we have that $(-a \mod b) = b - (a \mod b)$ if $(a \mod b) > 0$ and $(-a \mod b) = 0$ if $(a \mod b) = 0$. This allows us to conclude that $(a \mod b) + (-a \mod b) = b$ if $(a \mod b) > 0$ and $(a \mod b) + (-a \mod b) = 0$ if $(a \mod b) = 0$.

Inequalities under modulo can be written compactly: if $(a \mod b) > (c \mod b)$, we may equivalently write that $(a > c) \mod b$. Finally, we may freely add or subtract multiples of the modulus: $(a \mod b) = (a + 2b \mod b)$.

Values λ and μ. Notation defined and used in this section is summarized, for the reader's convenience, in Table 1.

The study of the schedule tardiness bound is based on two values, λ and μ, that will be extensively used in the rest of the paper, and are formally defined as follows (see Fig. 1 for an explicating example):

Definition 4 (*Values λ and μ*) For a given uniform instance (N, L, M, P), we define

$$\lambda = \left\lceil \frac{N}{M} \right\rceil L - P, \tag{4}$$

and

$$\mu = P - \left\lfloor \frac{N}{M} \right\rfloor L. \tag{5}$$

Intuitively, consider the first period and (according to the scheduling policy) assign jobs to processors following a round-robin policy. Some processors will be assigned $\left\lfloor \frac{N}{M} \right\rfloor$ jobs, the others $\left\lceil \frac{N}{M} \right\rceil$. Consider the latter processors and assume that $0 < \lambda, \mu \leq L$, then value λ denotes how much executing the $\left\lceil \frac{N}{M} \right\rceil$ jobs exceeds the length of the first period, and μ is the remaining time of the first period after the execution of $\left\lfloor \frac{N}{M} \right\rfloor$ jobs.

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

$$\lambda + \mu = \left\lceil \frac{N}{M} \right\rceil L - P + P - \left\lfloor \frac{N}{M} \right\rfloor L = \left(\left\lceil \frac{N}{M} \right\rceil - \left\lfloor \frac{N}{M} \right\rfloor \right) L = L, \tag{6}$$

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

Easy instances. There are some instances for which it is easy to state that the schedule tardiness is always zero because it is always possible to schedule all jobs of each period within the period. We will characterize these instances here and we

will no longer take them into consideration in the rest of the paper after this section. These instances are identified by the following conditions:

- If $(N \mod M) = 0$ we have $\left\lfloor \frac{N}{M} \right\rfloor = \left\lceil \frac{N}{M} \right\rceil = \frac{N}{M}$ (see Fig. 2a). In each period, each processor executes exactly $\frac{N}{M}$ jobs and terminates $\frac{N}{M}L \leq P$ time units after the beginning of the period, that is before the start of the following period. Note that all instances where $N = M$ fall into this category of the easy cases.
- If $\mu = 0$ then we have $P = \left\lfloor \frac{N}{M} \right\rfloor L$ (see Fig. 2b), implying $(N \mod M) = 0$, and the previous result applies also in this case. Since $P \geq \frac{N}{M}L \geq \left\lfloor \frac{N}{M} \right\rfloor L = P$, we have that $\frac{N}{M}L = \left\lfloor \frac{N}{M} \right\rfloor L$. The difference with the former case is that all available computation time is always used, i.e. $NL = MP$.
- If $(N \mod M) \neq 0$ and $\mu \geq L$ or if $\lambda \leq 0$ then, by definition of μ and λ, in each period it is always possible to assign $\left\lceil \frac{N}{M} \right\rceil$ jobs to $(N \mod M)$ processors and $\left\lfloor \frac{N}{M} \right\rfloor$ to the others, incurring in zero tardiness (see Fig. 2c).

Difficult instances. 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*.

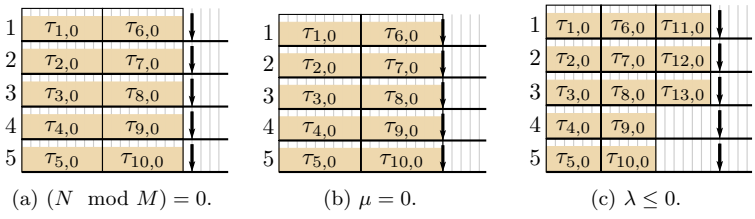
Lemma 1 *In all difficult instances, the following inequality holds:*

$$\frac{M}{N \mod M} \geq \frac{L}{\mu}. \tag{7}$$

Proof

$$\frac{M}{N \mod M} = \frac{M}{N - M\left\lfloor \frac{N}{M} \right\rfloor} \overset{(3)}{\geq} \frac{M}{M\frac{P}{L} - M\left\lfloor \frac{N}{M} \right\rfloor} = \frac{1}{\frac{P - L\left\lfloor \frac{N}{M} \right\rfloor}{L}} \overset{(5)}{=} \frac{L}{\mu}.$$

□



(a) $(N \mod M) = 0$. (b) $\mu = 0$. (c) $\lambda \leq 0$.

Fig. 2 Graphical representation of easy instances where only the first period is scheduled, the successive are identical. We can see that tardiness is always zero

We now introduce some more definitions that will be useful in the rest of the paper. We start by defining specific classes of processors. Intuitively, those processors with tardiness strictly smaller than μ, and those having one extra job to execute in a period with respect to the others.

Definition 5 (*Available processors*) We define A_j as the set of processors that in period j have tardiness lower than μ:

$$A_j = \left\{m \in \{1, \dots, M\} : R_{m,j} < \mu\right\}.$$

We shall call such processors *available*.[2]

Example 2 Referring to Fig. 1, all processors are available in period one. Indeed, processors 4 and 5 have tardiness equal to zero (and $0 < \mu$), and the tardiness of processors 1, 2, and 3 is equal to $\lambda = 4 \leq 5 = \mu$.

Definition 6 (*Selected and non-selected processor*) A processor that is assigned $\left\lceil \frac{N}{M} \right\rceil$ jobs in a given period is said to be *selected*, and the action of assigning such number of jobs to the processor is referred to as *selecting* the processor.

The set of processors that are selected in period j is denoted by S_j, and the set of *non-selected* processors is denoted by $\overline{S}_j = \{1, \dots, M\} \backslash S_j$.

Example 3 Referring to Fig. 1, we have $\left\lceil \frac{N}{M} \right\rceil = 3$. In the first period, processors 1, 2, and 3 are selected, as they are assigned 3 jobs. In the second period, the selected processors are 1, 4, and 5.

We now define a possible assignment of jobs to the processors that is admissible for a non-preemptive scheduler with uniform instances. As we will prove, this configuration is always applied by the scheduling policy and it characterizes the schedule tardiness.

Definition 7 (*Standard configuration*) We say that the *standard configuration* applies to a period j if, in the period, the following happens:

- exactly ($N \mod M$) processors are selected,
- no processor is assigned more than $\left\lceil \frac{N}{M} \right\rceil$ jobs of period j.

Remark 2 By a counting argument, we have that non-selected processors, under the standard configuration, are assigned exactly $\left\lfloor \frac{N}{M} \right\rfloor$.

[2] Note that the concept of available processors differs from the one we introduced in the conference version of this paper Buzzega et al. (2023), where the available processors were the ones without tardiness at the start of the period.

Definition 8 (*Standard configuration start*) We say that period j has a *standard configuration start* if and only if the two following conditions hold:

1. period j has tardiness bounded by L (i.e., $R_j < L$),
2. in period j there are at least $(N \mod M)$ available processors (i.e., $|A_j| \geq (N \mod M)$).

Example 4 Referring to Fig. 1, the standard configuration applies to period zero. Indeed, processors 1, 2, and 3 are selected and $(N \mod M) = 3$, and no processor is assigned more than $3 = \left\lceil \frac{N}{M} \right\rceil$ jobs. Moreover, period 1 has a standard configuration start, as $R_1 = 4 < 9 = L$ and all processors are available, as $R_1 = 4 < 5 = \mu$.

We now prove that we can ignore precedence constraints when scheduling the jobs in a period having tardiness bounded by the job length.

Lemma 2 *If $R_j < L$, precedence constraints of jobs of period $j-1$ over jobs of period j do not cause any idle time.*

Proof By time $t = (j-1)P + R_j$ all processors executed all jobs of previous periods, thus starting at time t, the jobs of period j are not subject to precedence constraints from earlier periods. Therefore, we concentrate on job assignments in the time interval $[(j-1)P..t-1]$, which, by hypothesis, lasts less than L time units. We show that the first job of period j to be assigned to any processor is not subject to precedence constraints.

As $R_j < L$, at the beginning of period j a processor might be executing its last job of period $j-1$. Consider the beginning of period j, we have two cases:

- **Case 1**: All the processors are executing their last job from period $(j-1)$; in period j each processor can be assigned the next job of the same task as soon as the previous one ends;
- **Case 2**: There are $n < M$ processor executing their last job from period $(j-1)$, and the same arguments as Case 1 can be applied. The remaining $M-n$ processors have tardiness equal to zero. Observe that there are $N-n$ jobs of period $j-1$ that ended their execution within period $j-1$, and do not pose precedence constraints on their corresponding jobs in period j. As $N-n \geq M-n$ (by hypothesis $N \geq M$), all the $M-n$ processors can start executing one job among these $N-n$, with no idle time.

□

5 Scheduling tardiness bound and hyper-period

In this section, we present the main result of the paper. We show that, for any uniform instance, the schedule tardiness is strictly bounded by L. The proof relies on Lemma 6 which, for the sake of presentation, will be proven later in Sect. 6. Specifically, Theorem 7 proves by induction that tardiness is bounded by the job

length. If in all periods preceding j we have a standard configuration start, then we can show that: (i) in period j tardiness is lower than L; (ii) by using Lemma 6, that period $j+1$ has enough available processors to cause another standard configuration start.

We conclude the section reporting results concerning the schedule hyperperiod. These are based on the results that appeared in Buzzega et al. (2023). The proofs will be then presented in Sect. 7.

5.1 Tardiness bound

The following Lemma provides a sufficient condition for the application of the standard configuration in a period.

Lemma 3 (Standard configuration) *Consider period $j \geq 0$, if j has a standard configuration start, then the standard configuration applies and the selected processors are available (i.e., $S_j \subseteq A_j$).*

Proof By hypothesis, for period j we have $L > R_j \geq 0$. We distinguish two cases:

Case 1: $R_j = 0$, i.e., the tardiness of period j is 0 (e.g. in period $j = 0$). Then, all processors have tardiness equal to zero as well, and the thesis follows by the pigeonhole principle (as in the first period in Fig. 1).

Case 2: $0 < R_j < L$, then all the jobs of period $j-1$ must have started their execution before the end of period $j-1$, and in period j, only the N jobs of period j must be scheduled.

By Lemma 2 precedence constraints cannot cause any idle time, so, by the work-conserving property of the scheduler, all processors are assigned $\left\lfloor \frac{N}{M} \right\rfloor$ jobs before the start of period $(j+1)$. The assignment of the last of these jobs is done before the end of the period, in particular by time t given by the sum of: (i) the end of period $j-1$, (ii) the tardiness of period j, and (iii) the time needed to execute the preceding $\left\lfloor \frac{N}{M} \right\rfloor - 1$ jobs. We have,

$$t = (j-1)P + R_j + \left(\left\lfloor \frac{N}{M} \right\rfloor - 1\right)L < (j-1)P + \left\lfloor \frac{N}{M} \right\rfloor L \leq (j-1)P + \frac{N}{M}L \overset{(3)}{\leq} jP,$$

i.e., t falls before the end of period P.

Moreover, the remaining $N - \left\lfloor \frac{N}{M} \right\rfloor M = (N \mod M)$ jobs are assigned to the processors with smaller tardiness at the beginning of period j, one each, as they are the first to complete their previous assignments. By hypothesis, there are at least $(N \mod M)$ available processors, and $(N \mod M)$ of these will also be selected.

□

Corollary 4 *Let $j \geq 0$, if period j has a standard configuration start, then in period j we have that*:

1. *selected processors are assigned* $\left\lceil \frac{N}{M} \right\rceil$ *jobs and increase their tardiness by* λ *in period* $j+1$, *i.e.*, $R_{m,j+1} = R_{m,j} + \lambda \quad \forall m \in S_j$;
2. *non-selected non-available processors are assigned* $\left\lfloor \frac{N}{M} \right\rfloor$ *jobs, i.e.*, $R_{m,j+1} = R_{m,j} - \mu \quad \forall m \in \overline{S}_j \backslash A_j$;
3. *non-selected available processors reset, i.e.*, $R_{m,j+1} = 0 \quad \forall m \in \overline{S}_j \bigcap A_j$.

Proof The proof follows from the definitions of λ and μ used in equations analogous to those of the proof of Lemma 3. Let m be a processor and t_m the time in which it ends the execution of the jobs assigned to it in period j. We consider the three cases separately:

1. Processor m is selected, then it is assigned $\left\lceil \frac{N}{M} \right\rceil$ jobs (by definition) and
$$t_m = (j-1)P + R_{m,j} + \left\lceil \frac{N}{M} \right\rceil L \overset{(4)}{=} (j-1)P + R_{m,j} + P + \lambda = jP + R_{m,j} + \lambda,$$
and thus we have $R_{m,j+1} = R_{m,j} + \lambda$.
2. Processor m is not selected and not available, then it is assigned $\left\lfloor \frac{N}{M} \right\rfloor$ jobs (by Remark 2) and
$$t_m = (j-1)P + R_{m,j} + \left\lfloor \frac{N}{M} \right\rfloor L \overset{(5)}{=} (j-1)P + R_{m,j} + P - \mu = jP + R_{m,j} - \mu,$$
and thus we have $R_{m,j+1} = R_{m,j} - \mu$.
3. Processor m is available and not selected, then it is assigned $\left\lfloor \frac{N}{M} \right\rfloor$ jobs (by Remark 2), $R_{m,j} < \mu$ and
$$t_m = jP + R_{m,j} - \mu < jP,$$
and thus we have $R_{m,j+1} = 0$.

□

The following corollary and lemma are fundamental for the inductive proof of the main theorem of the paper. The proof of Lemma 6 will be the focus of the next section.

Corollary 5 (Inductive step 1) *Let* $j \geq 0$, *if period* j *has a standard configuration start, then the tardiness of period* $j+1$ *is also strictly smaller than* L, *i.e.*, $R_{j+1} < L$.

Proof Processor m that is selected in period j had tardiness $R_{m,j} < \mu$ (because it is also available) and incurs a tardiness in period $j+1$ equal to $R_{m,j+1} = R_{m,j} + \lambda < \mu + \lambda = L$. The other processors are not selected and do not increase their tardiness in period $j+1$ w.r.t. period j. □

Lemma 6 (*Inductive step* 2) *Let* $j \geq 0$, *If each period* $k \in [0..j]$ *has a standard configuration start, then there are at least* $(N \mod M)$ *available processors also in period* $j+1$ *(i.e.,* $|A_{j+1}| \geq (N \mod M)$*).*

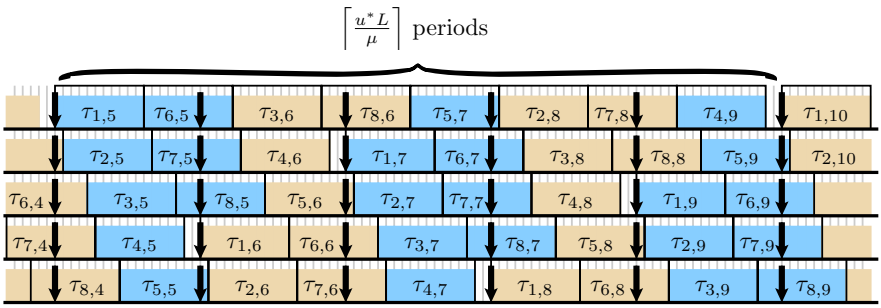


Fig. 3 Scheduled example of the instance $(N, L, M, P) = (8, 11, 5, 18)$ in which $\mu = 18 - 11 = 7$, $u^* = 3$, so by Theorem 10 the hyperperiod length is 6. It is evident in the figure that the tardiness distribution at the beginning of period 5 is equivalent to the one at the beginning of period 12; the periodicity follows by the deterministic and work-conserving nature of the scheduler

We are now ready to state and prove that, for any period $j \geq 0$, the period tardiness is strictly bounded by L.

Theorem 7 (Schedule Tardiness Upper Bound) *Consider any uniform instance* (N, L, M, P)*, and let* T *be the schedule tardiness. Then we have* $T < L$.

Proof The thesis is true for easy instances, because their schedule tardiness is always zero, as shown in Sect. 4. Let us now consider difficult instances. The proof is by induction, and we will show that any period $j \geq 0$ has a standard configuration start. The proof then follows from the first condition of standard configuration start, because T is the maximum over all periods tardiness.

Base of the induction: By definition period zero has a standard configuration start: $R_0 = 0$ and $|A_0| = M$.

Inductive step: Assume each period $k \in [0..j]$ has a standard configuration start. Then, in period $j + 1$ the following hold: (1) by Corollary 5, we have that $R_{j+1} < L$ and, (2) by Lemma 6 we have that $|A_{j+1}| \geq (N \mod M)$. This means that also period $j + 1$ has a standard configuration start. □

Corollary 8 *The* $T < L$ *bound for the scheduling tardiness of uniform instances is tight*.

Proof For the class of uniform instances given by $(N, L, M, P) = (k + 1, k - 1, k, k)$, $k \in \mathbb{N}^+$ and $k > 1$, the tardiness of period one is $\lambda = \left\lceil \frac{k+1}{k} \right\rceil (k - 1) - k = 2(k - 1) - k = k - 2 = L - 1$. □

Corollary 9 *Given a uniform instance* (N, L, M, P)*, in any period* j*, the number of available processors is always at least* $(N \mod M)$*, i.e.*:

$$|A_j| \geq (N \mod M) \quad \forall j \geq 0.$$

5.2 Hyper-period length

We conclude this section by reporting the result concerning the length of the schedule hyper-period. We prove that the schedule is periodic and characterize the length of the hyper-period. An example of hyper-period is shown in Fig 3.

We make use of the value u^*, first presented in Buzzega et al. (2023), to partition the set of uniform instances into classes. Here, instead, we show that it can be used to compute the hyper-period length and the exact schedule tardiness value. All proofs are presented in Sect. 7, where we also report an upper bound for u^*, and we show that it is computable.

Theorem 10 (Hyper-period length) *Given a difficult instance* (N, L, M, P) *, let* $u \in \mathbb{N}^+$ *be the maximum number of times a processor is selected before resetting. Then the hyper-period length is equal to* $\left\lceil u\frac{L}{\mu} \right\rceil$.

Definition 9 Given a difficult instance (N, L, M, P), we define the value

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

Theorem 11 *For any difficult instance* (N, L, M, P) *,* u^* *is the maximum number of times any processor is selected before resetting.*

Corollary 12 (*Exact maximum tardiness value*) *Consider a uniform instance* (N, L, M, P). *The maximum tardiness* T *is*

$$T = \max(0, \lambda + \max_{0 \leq i < u^*} i\lambda \mod \mu).$$

6 About available processors

The proof of Theorem 7 relies on Lemma 6, which is not difficult to state and understand. Unfortunately, the proof of the Lemma is not trivial at all; on the contrary, it is the most demanding task of the paper. To prove the existence of a set of available processors in a given period under a standard configuration regime, we study how the tardiness of the processors in this set decreases in the preceding periods. By going backward to the first periods we are able to show that the cardinality of the set is at least $(N \mod M)$.

Example 5 In Fig. 3, in period 8 there are 4 available processors, namely the ones corresponding to the first, the second, the third, and the last row. The first was previously selected in period 6 and had to decrease its tardiness in period 7 to become available in period 8. The second and third were selected in period 7, but their tardiness was still below μ. Finally, the last one was available and not selected in period

7, so its tardiness reset. In this case, the number of available processors in period 8 is $4 \geq (N \mod M)$. Lemma 6 proves that, if its hypotheses are met, this is the case in any period.

Looking at the examples in the figures and knowing that there is a hyper-period, it might seem easy to characterize the processor's tardiness, because it shows some regularity. However, the "type" of regularity might vary according to the relations between the values that characterize the instance, and its study proves to be not trivial.

The notation defined in this section is reported, for the reader's convenience, in Table 2).

6.1 Yet some more notation and preliminaries

We give two definitions that allow us to characterize with a finer grain the tardiness values of available processors.

Definition 10 Let us define two disjoint sets of integers:

- $W^{>} = \{(i\lambda) \mod \mu : (i\lambda > (i+1)\lambda) \mod \mu, i \in \mathbb{Z}\}$;
- $W^{\leq} = \{(i\lambda) \mod \mu : (i\lambda \leq (i+1)\lambda) \mod \mu, i \in \mathbb{Z}\}$.

Definition 11 We define two sets of processors:

- $A_j^{>}$ is the set of processors that start period j with a tardiness value that belongs to the set $W^{>}$;
- $A_j^{\leq}$ is the set of processors that start period j with a tardiness value that belongs to the set $W^{\leq}$.

Remark 3 Observe that sets $A_j^{>}$ and $A_j^{\leq}$ are always disjoint because $W^{>}$ and $W^{\leq}$ are. Moreover, as the values in $W^{>}$ and $W^{\leq}$ are smaller than μ, we have that processors in $A_j^{>}$ and $A_j^{\leq}$ are available, i.e., $A_j^{>} \cup A_j^{\leq} \subseteq A_j$.

Table 2 Helper symbols

$W^{>}$	Set of tardiness values $(i\lambda \mod \mu)$ greater than their next multiple in modulo μ
$W^{\leq}$	Set of tardiness values $(i\lambda \mod \mu)$ smaller than their next multiple in modulo μ
$A_j^{>}$	Set of processors with tardiness in $W^{>}$ in period j
$A_j^{\leq}$	Set of processors with tardiness in $W^{\leq}$ in period j
$\pi_{m,j}$	Position of processor m in period j
Λ_i	Tardiness value equal to $\lambda + (i\lambda \mod \mu)$
q_i	Cooldown time for a processor with tardiness Λ_i
$q_{>}$	Cooldown time for a selected processor with tardiness in $W^{>}$, equal to $\lfloor L/\mu \rfloor$
$q_{\leq}$	Cooldown time for a selected processor with tardiness in $W^{\leq}$, equal to $\lfloor L/\mu \rfloor - 1$

Remark 4 Note that if $(\lambda \mod \mu) = 0$, then $(i\lambda \mod \mu) = 0 = ((i+1)\lambda \mod \mu)$ for all $i \in \mathbb{Z}$. Thus, $(i\lambda \mod \mu) \in W^{\leq}$ for all $i \in \mathbb{Z}$ and $A_j^{>}$ is empty.

The same holds if $(L \mod \mu) = 0$, because $L = \lambda + \mu$ for difficult instances, which implies $(\lambda \mod \mu) = 0$.

Remark 5 Given period j, if the tardiness $R_{m,j}$ of an available processor $m \in A_j$ can be expressed as a multiple of λ minus a multiple of μ, then $R_{m,j}$ belongs to $A_j^{>}$ or $A_j^{\leq}$.

Indeed, there exists $i, h \geq 0$ such that $R_{m,j} = i\lambda - h\mu$ and, as j is available, then $R_{m,j} < \mu$. This implies that $R_{m,j} = (i\lambda \mod \mu)$ and thus that it belongs to $A_j^{>} \cup A_j^{\leq}$.

The following lemma ties the order relation occurring between $i\lambda$ and $-\lambda$, modulo μ, to the membership of $(i\lambda \mod \mu)$ to either $W^{>}$ or $W^{\leq}$. See also Fig. 4 for a visual intuition of the statement of the lemma.

Lemma 13 *If* $(\lambda \mod \mu > 0)$ *then*

$$(i\lambda \mod \mu) \in W^{>} \Leftrightarrow (i\lambda \geq -\lambda) \mod \mu \quad i \in \mathbb{Z}; \tag{8}$$

dually

$$(i\lambda \mod \mu) \in W^{\leq} \Leftrightarrow (i\lambda < -\lambda) \mod \mu \quad i \in \mathbb{Z}. \tag{9}$$

Proof Let us first consider statement (8).

($\Leftarrow$) By hypothesis, we have that

$$(i\lambda \mod \mu) + (\lambda \mod \mu) \geq ((-\lambda) \mod \mu) + (\lambda \mod \mu) = \mu,$$

so $\mu \leq (i\lambda \mod \mu) + (\lambda \mod \mu) < 2\mu$ and

$$\begin{aligned}(i+1)\lambda \mod \mu &= ((i\lambda \mod \mu) + (\lambda \mod \mu)) \mod \mu \\ &= (i\lambda \mod \mu) + \underbrace{(\lambda \mod \mu) - \mu}_{<0} \\ &< i\lambda \mod \mu.\end{aligned}$$

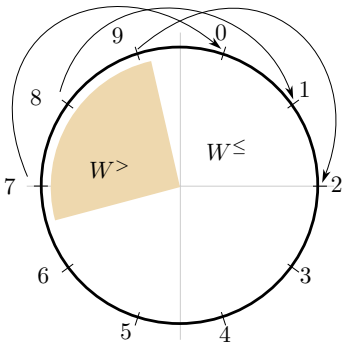


Fig. 4 Example related to Definitions 10 and 11: Tardiness values distribution for $(\lambda \mod \mu) = 3$ and $\mu = 10$. Arrows point to the next multiple of λ in modulo μ. Values of set $W^{>}$ are greater or equal to $((-\lambda) \mod \mu) = 7$

By definition, this implies that $(i\lambda \mod \mu) \in W^{>}$.

($\Rightarrow$) Let i be an integer such that $(i\lambda > (i+1)\lambda) \mod \mu$. This is only if $(i\lambda \mod \mu) + (\lambda \mod \mu) \geq \mu$, or, equivalently, $(i\lambda \mod \mu) \geq \mu - (\lambda \mod \mu) = ((-\lambda) \mod \mu)$.

For the statement (9), since $(\lambda \mod \mu) > 0$, $W^{\leq}$ and $W^{>}$ form a partition of the set of multiples of λ modulo μ. Thus, by statement (8), we have that $(i\lambda \mod \mu) \in W^{\leq} \Leftrightarrow (i\lambda \mod \mu) < ((-\lambda) \mod \mu)$. □

Example 6 In Fig 5 we can see that processor 1 in period 2 has tardiness $R_{1,2} = (\lambda \mod \mu) = 2 \geq (-\lambda \mod \mu) = 1$. Indeed we have that $2 > (2\lambda \mod \mu) = 1$, as Lemma 13 predicts.

6.2 Position of processors

We now introduce the concept of position of processors in a given period. The position depends on the number of processors in the period having tardiness smaller or equal to the given processor. Position is defined for all processors, independently from the fact of being selected or not, being available or not.

Let $b(m_1, m_2)$ be a tie-break function that, for any two idle processors m_1, m_2, is true if a job is assigned to processor m_1 over processor m_2, false otherwise. For example, if a scheduler prioritizes lower index processors, b is $<$.

Definition 12 (*Processor position*) We define $\pi_{m,j}$ as the *position* of processor m in period j, which corresponds to the number of processors with tardiness smaller than the tardiness of m in period j plus the number of processors that have the same tardiness of m and are prioritized by the tie-breaking function, i.e.,

$$\pi_{m,j} = \left|\left\{m' \in [1..M] : R_{m',j} < R_{m,j}\right\}\right| + \left|\left\{m' \in [1..M] : R_{m',j} = R_{m,j} \wedge b(m', m)\right\}\right|.$$

Observe that positions range from zero to $M - 1$ and processors have distinct positions.

Remark 6 As we have seen in the proof of Lemma 3, in the standard configuration the selected processors are the ones with the lowest tardiness. Since position

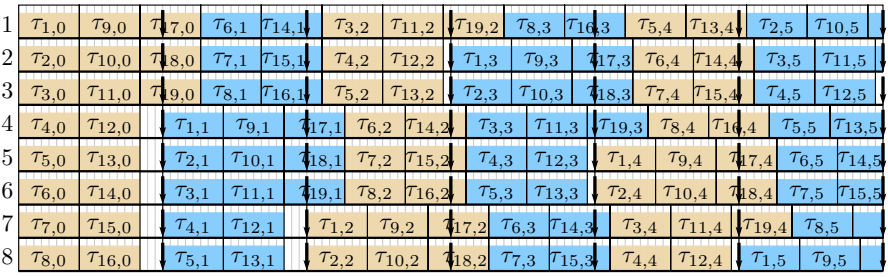


Fig. 5 Example: first periods of the schedule of the instance $(N, L, M, P) = (19, 8, 8, 19)$, in which $(N \mod M) = 3$, $\lambda = 5$ and $\mu = 3$

encodes both tardiness relative to other processors and the tie-breaking policy of the scheduler, in each period where the standard configuration applies selected processors are the $(N \mod M)$ with the lowest position.

Example 7 Let us consider Fig. 5 and assume that $<$ is the tie-breaking function b. In period 4, we see that processor 5 is in position 0, processor 6 is in position 1 (because of the tie-break), and processor 1 is in position 4.

Lemma 14 (*Position change in standard configuration*) *Consider a difficult instance* (N, L, M, P). *If period* $j \geq 0$ *has a standard configuration start, then selected processors increase their position by* $M - (N \mod M)$ *positions, while non-selected ones decrease it by* $(N \mod M)$ *i.e.:*

$$\pi_{m,j+1} = \begin{cases} \pi_{m,j} + M - (N \mod M) < M, & \text{if } m \in S_j, \\ \pi_{m,j} - (N \mod M) \geq 0 & \text{if } m \notin S_j \end{cases}$$

Example 8 Let us consider Fig. 5 again, in which $(N \mod M) = (19 \mod 8) = 3$ and $M - (N \mod M) = 5$. In period 4, processors 5 and 6 are selected and increase their position in period 5 from 0 and 1 to 5 and 6, respectively. Processor 1 is not selected in period 4 and decreases its position in period 5 from 4 to 1.

Proof By Lemma 3 the standard configuration applies in period j, so, by Remark 6 non-selected processors have position greater or equal to $(N \mod M)$. Let $m \in S_j$ be a selected processor, and $\overline{m} \in \overline{S_j}$ be a non-selected one, thus $0 \leq \pi_{m,j} < (N \mod M) \leq \pi_{\overline{m},j} < M$. We show that, in period $j+1$ the situation flips and we have that $\pi_{\overline{m},j+1} < \pi_{m,j+1}$. Indeed, by hypothesis, Corollary 4 applies and

$$R_{\overline{m},j+1} = R_{\overline{m},j} - \mu < L - \mu \overset{(6)}{=} L - \mu = \lambda \leq R_{m,j} + \lambda = R_{m,j+1}.$$

To conclude the proof, we observe that, by the definition of standard configuration, $|S_j| = (N \mod M)$ and $|\overline{S_j}| = M - (N \mod M)$. □

Lemma 15 *Consider any difficult instance* (N, L, M, P) *such that* $\left\lfloor \frac{M}{N \mod M} \right\rfloor = \left\lfloor \frac{L}{\mu} \right\rfloor$ *and period* $j \geq 0$. *If each period in* $[0..j-1]$ *has a standard configuration start, then each processor* $m \in A_j^{>}$ *has position*

$$\pi_{m,j} \geq (-M) \mod (N \mod M).$$

Proof There are three cases:

Case 1: $(M \mod (N \mod M) = 0)$. Then also $(-M) \mod (N \mod M) = 0$ and the thesis holds because positions are non-negative.

Case 2: $(\lambda \mod \mu) = 0$. By Remark 4, then $A_j^{>} = \emptyset$ and the thesis holds.

Case 3: $(M \mod (N \mod M) > 0)$ and $(\lambda \mod \mu) > 0$.

By Lemma 3 and Lemma 2 in all periods $[0..j-1]$ the standard configuration applies and precedence constraints cause no idle time. As $m \in A_j^{>}$ then its tardiness

can be expressed as $R_{m,j} = ((i\lambda) \mod \mu) \in W^{>}$ for some $0 < i < \frac{\text{lcm}(\lambda,\mu)}{\lambda}$. Indeed, when $i = \frac{\text{lcm}(\lambda,\mu)}{\lambda}$, $i\lambda \mod \mu = 0$ and $R_{m,j} \in W^{\leq}$.

By Corollary 4, m must have been selected i times (increasing each time its tardiness by λ w.r.t. the previous period) and non-selected h times (decreasing its tardiness by μ each time w.r.t the previous period), without resetting. As in period j processor m is in $A_j^{>}$, then $0 < R_{m,j} < \mu$, and the number h of decreases in tardiness must compensate for a total increase of $i\lambda$. Thus, $h = \left\lfloor \frac{i\lambda}{\mu} \right\rfloor$.

Assume that the most recent period in which processor m is selected after a reset is $k \in [0..j]$. By Lemma 14, we can state that, by period j the position of processor m increased by $(M - (N \mod M))$ positions each period it was selected, and decreased of $(N \mod M)$ positions each period it was not selected without resetting:

$$\pi_{m,j} = \pi_{m,k-1} + i(M - (N \mod M)) - h(N \mod M).$$

As $\pi_{m,k-1} \geq 0$, the thesis follows by calculation:

$$\begin{aligned}
\pi_{m,j} &\geq iM - i(N \mod M) - \left\lfloor \frac{i\lambda}{\mu} \right\rfloor (N \mod M) \\
&\overset{(6)}{=} iM - \left(i\frac{L}{\mu} - \frac{(i\lambda) \mod \mu}{\mu} \right) (N \mod M) \\
&\overset{(8)}{\geq} iM - \left(i\frac{L}{\mu} + \frac{(-\lambda) \mod \mu}{\mu} \right) (N \mod M) \\
&= i\left(M - \frac{L}{\mu}(N \mod M) \right) + \frac{\mu - (L \mod \mu)}{\mu} (N \mod M) \\
&\overset{(7)}{\geq} \left(1 - \left(\frac{L}{\mu} - \left\lfloor \frac{L}{\mu} \right\rfloor \right) \right) (N \mod M) \\
&\overset{(7)}{\geq} \left(1 - \left(\frac{M}{N \mod M} - \left\lfloor \frac{M}{N \mod M} \right\rfloor \right) \right) (N \mod M) \\
&= \left(1 - \frac{(M) \mod (N \mod M)}{N \mod M} \right) (N \mod M) \\
&= (-M) \mod (N \mod M).
\end{aligned}$$

□

6.3 Cooldown time

We now concentrate on what is defined as the cooldown time, that is the minimum number of periods needed by a selected processor to become available again, under a standard configuration regime. We will show that this value solely depends on whether the tardiness of the processor belongs to either $W^{>}$ or $W^{\leq}$.

Definition 13 (*Cooldown time*) Let us consider a processor with tardiness $\Lambda_i = \lambda + (i\lambda \mod \mu)$ in some period k. The *cooldown time* q_i is the number of consecutive periods after k in which, under a standard configuration regime, that are necessary for the processor to reduce its tardiness to a value smaller than μ, without being selected.

Lemma 16 *Consider a processor with tardiness* $\Lambda_i = \lambda + (i\lambda \mod \mu)$ *in some period* k. *Assume that the processor is not selected for* q_i *consecutive periods, and that period* $k + q_i$ *is the first in which the processor tardiness is smaller than* μ. *Then, we have*

$$q_i = \frac{\lambda + (i\lambda \mod \mu) - ((i+1)\lambda \mod \mu)}{\mu}. \tag{10}$$

Proof In each period in which the processor is not selected, it reduces its tardiness by μ. Thus, to reduce the tardiness to a value smaller than μ it needs exactly $\lfloor \Lambda_i/\mu \rfloor$ consecutive periods. Therefore, using the definitions of integer division and modulo, by calculation we have:

$$\begin{aligned} q_i &= \left\lfloor \frac{\lambda + (i\lambda \mod \mu)}{\mu} \right\rfloor \\ &= \frac{\lambda + (i\lambda \mod \mu) - ((\lambda + (i\lambda \mod \mu)) \mod \mu)}{\mu} \\ &= \frac{\lambda + (i\lambda \mod \mu) - ((i+1)\lambda \mod \mu)}{\mu}. \end{aligned}$$

□

Example 9 In Fig. 5 processor 1 in period 3 has tardiness $R_{1,3} = \Lambda_1 = \lambda + (1\lambda \mod \mu) = 7$. As we can see, the standard configuration applies in the first 5 periods, so processor 1 needs $q_1 = \left\lfloor \frac{7}{3} \right\rfloor = 2$ periods to become available again. Indeed $R_{1,3+q_1} = R_{1,5} = 1 < \mu$. Note that in periods 3 and 4, processor 1 is not yet available.

Lemma 17 *Consider a difficult instance* (N, L, M, P), *a period* $j \geq 0$, *and a processor* m *in period* j *with tardiness* $R_{m,j} = \Lambda_i = \lambda + (i\lambda \mod \mu)$ *for some* $i \geq 0$. *If each period* $h \in [j..j + q_i - 1]$ *has a standard configuration start and* m *is never available (i.e.,* $R_{m,h} \geq \mu$*), then* m *will never be selected in the interval* $[j..j + q_i - 1]$ *and*

$$R_{m,(j+q_i)} = ((i+1)\lambda \mod \mu).$$

Proof Processor m will never be selected in $[j..j + q_i - 1]$ because its tardiness is not $< \mu$, by Lemma 3. Moreover, by Corollary 4, the processor tardiness is decreased by μ, w.r.t the previous period, in each of the q_i periods in the interval. Thus, at period $j + q_i$ the tardiness of processor m is:

$$R_{m,(j+q_i)} = \lambda + (i\lambda \mod \mu) - \mu q_i = ((i+1)\lambda \mod \mu), \tag{11}$$

where we used (10). □

Remark 7 Under the same hypothesis of Lemma 17, we can show that q_i may only assume two values, depending on the value of i:

$$q_i = \begin{cases} q_> = \left\lfloor \frac{L}{\mu} \right\rfloor & \text{if } (i\lambda \mod \mu) \in W^>, \\ q_\leq = \left\lfloor \frac{L}{\mu} \right\rfloor - 1 & \text{if } (i\lambda \mod \mu) \in W^\leq. \end{cases}$$

Remembering that by Remark 1 we have that $\lambda + \mu = L$, let us consider the two cases separately:

- If $(i\lambda \mod \mu) \in W^>$,

$$\begin{aligned} &\Rightarrow (i\lambda > (i+1)\lambda) \mod \mu, \text{ by definition of } W^> \\ &\Rightarrow ((i+1)\lambda \mod \mu) = (i\lambda \mod \mu) + (\lambda \mod \mu) - \mu \\ &\Rightarrow q_i = \frac{\lambda + (i\lambda \mod \mu) - (i\lambda \mod \mu) - (\lambda \mod \mu) + \mu}{\mu} \\ &\quad = \frac{\lambda + \mu - L \mod \mu}{\mu} = \frac{L - L \mod \mu}{\mu} = \left\lfloor \frac{L}{\mu} \right\rfloor \end{aligned}$$

- If $(i\lambda \mod \mu) \in W^\leq$

$$\begin{aligned} &\Rightarrow (i\lambda \leq (i+1)\lambda) \mod \mu, \text{ by definition of } W^\leq \\ &\Rightarrow ((i+1)\lambda) \mod \mu = (i\lambda \mod \mu) + (\lambda \mod \mu) \\ &\Rightarrow q_i = \frac{\lambda + (i\lambda \mod \mu) - (i\lambda \mod \mu) - (\lambda \mod \mu)}{\mu} \\ &\quad = \frac{\lambda - L \mod \mu}{\mu} = \frac{L - L \mod \mu}{\mu} - 1 = \left\lfloor \frac{L}{\mu} \right\rfloor - 1 \end{aligned}$$

6.4 Back to available processors

In this section, we finally prove Lemma 6.

In the next proofs, in a period $k > q_>$, we focus on the following set of selected processors (that will be denoted with $S_{k,1}, S_{k,2}$ and $S_{k,3}$, respectively, for better readability), under the standard configuration regime:

- Processors available but not selected in period k, i.e., $A_k \bigcap \overline{S}_k = S_{k,1}$;
- Non-available processors that will become available in period $k+1$. In particular, depending on their cooldown time value $q_i \in \{q_\leq, q_>\}$, we have:

– processors available and selected in period $k - q_>$, i.e., $A^>_{k-q_>} \bigcap S_{k-q_>} = S_{k,2}$;
– processors available and selected in period $k - q_\leq$, i.e., $A^\leq_{k-q_\leq} \bigcap S_{k-q_\leq} = S_{k,3}$.

Remark 8 Observe that the three sets are disjoint. Processors in $S_{k,1}$ are available in period k, while processors in $S_{k,2}$ and $S_{k,3}$ are not available.

Processors in $S_{k,2}$ can not be in $S_{k,3}$ (and vice versa) because their cooldown time differs by one period. Indeed, processors of set $S_{k,2}$ are not available in period $(k - q_\leq) \leq k$, while processors of set $S_{k,3}$ are.

Remark 9 The processors in $S_{k,1} \cup S_{k,2} \cup S_{k,3}$ are those that will be available in period $k + 1$, i.e., are subsets of A_{k+1}. Indeed, by Corollary 4:

- Processors in $S_{k,1}$ have tardiness strictly smaller than μ and are not selected in period k, thus they will decrease their tardiness in period $k + 1$ by μ, resetting.
- Processors in $S_{k,2}$ and $S_{k,3}$ will complete their cooldown time in period $k + 1$ becoming available.
- All other processors in k have tardiness of at least μ and are at least two periods away from their cooldown time, thus will not be available at period $k + 1$.

Lemma 18 *Let $j \geq q_>$. If for each $k \in [0..j]$ the following holds:*

1. *period k has a standard configuration start,*
2. *the tardiness of each processor in the period is given by a multiple of λ minus a multiple of μ , i.e., $\forall m \in [1..M], \exists i, h \geq 0 : R_{m,k} = i\lambda - h\mu$,*

then, the union of sets $S_{j,1}, S_{j,2}$ and $S_{j,3}$ contains at least $(N \mod M)$ processors, i.e.,

$$\left| \underbrace{\left(A_j \bigcap \overline{S}_j\right)}_{S_{j,1}} \bigcup \underbrace{\left(A^>_{j-q_>} \bigcap S_{j-q_>}\right)}_{S_{j,2}} \bigcup \underbrace{\left(A^\leq_{j-q_\leq} \bigcap S_{j-q_\leq}\right)}_{S_{j,3}} \right| \geq N \mod M.$$

Proof By Lemma 3, in each period $k \in [0..j]$ the standard configuration applies and the $(N \mod M)$ selected processors are also available processors (i.e., $|S_k| = (N \mod M)$ and $S_k \subseteq A_k$). Moreover, by Lemma 2 precedence constraints cause no idle time.

By hypothesis, all processors in period k have tardiness values expressed as multiples of λ minus multiples of μ, thus this applies also to all available processors, and by Remark 5 we have that the tardiness of any available processor belongs to $A^>_k \bigcup A^\leq_k = A_k$. As processors are selected among available ones, we have:

$$|A^>_k \bigcap S_k| + |A^\leq_k \bigcap S_k| = |S_k| = (N \mod M), \tag{12}$$

because $A^>_k$ and $A^\leq_k$ are disjoint.

We distinguish two cases:

Case 1: If $(L \mod \mu) = 0$, then by Remark 4 we have that, $A_k^{>} = \emptyset$. By eq. (12) for $k = j - q_{\leq}$, we have that $|S_{j,3}| = |A_{j-q_{\leq}}^{\leq} \bigcap S_{j-q_{\leq}}| = (N \mod M)$, which implies the thesis.

Case 2: $(L \mod \mu) > 0$. Let $g \in [q_{>}..j]$. By Remark 9 we have that the available processors in A_g are those in the $S_{g-1,.}s$ of the previous period:

$$A_g = \bigcup_{i=1}^{3} S_{g-1,i} \text{ and } |A_g| = \sum_{i=1}^{3} |S_{g-1,i}|, \tag{13}$$

because the sets are disjoint. Processors in $S_{g,1}$ are those available in period g but not selected in g, thus

$$|S_{g,1}| = |A_g \bigcap \overline{S}_g| = |A_g| - |S_g| \overset{(13)}{=} \sum_{i=1}^{3} |S_{g-1,i}| - |S_g|. \tag{14}$$

Let's start evaluating the sum of cardinalities of $S_{j,1}$ and $S_{j,2}$. We have that:

$$|S_{j,1}| + |S_{j,2}| \overset{(14)}{=} |S_{j-1,1}| + |S_{j-1,2}| + \left(|S_{j-1,3}| + |S_{j,2}|\right) - |S_j|. \tag{15}$$

Observe that, by definition $q_{>} = q_{\leq} + 1$, implying that $j - q_{>} = j - 1 - q_{\leq}$, and thus

$$S_{j,2} = A_{j-q_{>}}^{>} \bigcap S_{j-q_{>}} = A_{j-1-q_{\leq}}^{>} \bigcap S_{j-1-q_{>}}. \tag{16}$$

And using equation (12) for $k = j - 1 - q_{\leq}$, this implies that

$$|S_{j-1,3}| + |S_{j,2}| = |A_{j-1-q_{\leq}}^{\leq} \bigcap S_{j-1-q_{\leq}}| + |A_{j-1-q_{\leq}}^{>} \bigcap S_{j-1-q_{\leq}}| = |S_{j-1-q_{\leq}}|.$$

Using (15) in (16) we have that

$$|S_{j,1}| + |S_{j,2}| = |S_{j-1,1}| + |S_{j-1,2}| + \underbrace{|S_{j-1-q_{\leq}}| - |S_j|}_{=0} = |S_{j-1,1}| + |S_{j-1,2}|, \tag{17}$$

because the number of selected processors, under the current hypothesis, is always equal to $(N \mod M)$, independently of the index of the period.

Repeating these substitutions multiple times we get that

$$|S_{j,1}| + |S_{j,2}| = \underbrace{|S_{q_{>},1}|}_{=0} + |S_{q_{>},2}| = |A_{q_{>}}| - (N \mod M).$$

Indeed, observe that $S_{q_{>},1} = A_0^{>} \bigcap S_0$ and that in period zero $A_0^{>}$ is empty because all processors have tardiness equal to zero and $0 \in W^{\leq}$. Therefore, $|S_{q_{>},1}| = 0$. Moreover, $S_{q_{>},2}$ is the set $A_{q_{>}}$ of processors available in period $q_{>}$ that have not been selected, therefore $S_{q_{>},2} = A_{q_{>}} \setminus S_{q_{>}}$.

Moreover, in the interval going from period zero to period $q_{>} - 1$, at most $q_{>}(N \mod M)$ distinct processors have been selected, and so at least $M - q_{>}(N \mod M)$ processors have never been selected. In addition, the processors

that have been selected in period zero, have tardiness $\Lambda_0 = \lambda + (0\lambda \mod \mu)$ in period one. Since $(0\lambda \mod \mu) \in W^{\leq}$ their cooldown time is $q_{\leq}$ and, by Lemma 17, in period $1 + q_{\leq} = q_{>}$ these processors are available again. Hence $|A_{q_{>}}| \geq M - q_{>}(N \mod M) + (N \mod M)$.

Summing up:

$$\begin{aligned}
&|S_{j,1}| + |S_{j,2}| \\
&\quad = |A_{q_{>}}| - (N \mod M) \\
&\quad = M - q_{>}(N \mod M) + (N \mod M) - (N \mod M) \\
&\quad = \left(\left\lfloor \frac{M}{N \mod M} \right\rfloor - \left\lfloor \frac{L}{\mu} \right\rfloor \right)(N \mod M) + (M \mod (N \mod M)).
\end{aligned}$$

We further distinguish two more cases:

Case 2.1: $\left\lfloor \frac{M}{N \mod M} \right\rfloor > \left\lfloor \frac{L}{\mu} \right\rfloor$. Then $|S_{j,1}| + |S_{j,2}| \geq (N \mod M)$, because the two sets are disjoint by Remark 8, and we have the thesis.

Case 2.2: $\left\lfloor \frac{M}{N \mod M} \right\rfloor = \left\lfloor \frac{L}{\mu} \right\rfloor$ and we have that $|S_{j,1}| + |S_{j,2}| = (M \mod (N \mod M))$. By Lemma 15 available processors $m_{>} \in A^{>}_{j-q_{\leq}}$ have position $\pi_{m_{>},j-q_{\leq}} \geq (-M) \mod (N \mod M)$. Therefore, lower positions must be occupied by available processors with tardiness in $W^{\leq}$ (i.e., in $A^{\leq}_{j-q_{\leq}}$), and at least $((-M) \mod (N \mod M))$ of these processors must be selected in period $j - q_{\leq}$ because $((-M) \mod (N \mod M)) < N \mod M$. Therefore,

$$|S_{j,3}| = |A^{\leq}_{j-q_{\leq}} \bigcap S_{j-q_{\leq}}| \geq (-M) \mod (N \mod M),$$

and, by modulo proprieties,

$$\begin{aligned}
|S_{j,1}| + |S_{j,2}| + |S_{j,3}| &\geq (M \mod (N \mod M)) + ((-M) \mod (N \mod M)) \\
&= N \mod M,
\end{aligned}$$

which finally implies the thesis because the sets are disjoint, by Remark 8. □

We are finally ready to prove Lemma 6, whose statement is here reported for the reader's convenience.

Lemma 6 (Inductive step 2) *Let $j \geq 0$, If each period $k \in [0..j]$ has a standard configuration start, then there are at least $(N \mod M)$ available processors also in period $j + 1$ (i.e., $|A_{j+1}| \geq (N \mod M)$).*

Proof Consider period $j \geq 0$. By hypothesis and by Lemma 3 observe that the standard configuration applies in all periods up to period j included. Now we distinguish three cases:

Case 1: $j \in \left[0..\left\lfloor \frac{M}{N \mod M} \right\rfloor - 2\right]$.

By Corollary 4 only the $(N \mod M)$ selected processors increase their tardiness in each period $k \in [0..j]$. As exactly $(N \mod M)$ processors are selected in

each period, by period j at most $(j+1)\cdot(N \mod M)$ distinct processors have been selected, and the number of those that have never been selected up to period $j+1$ is at least

$$M-(j+1)(N \mod M) \geq M-\left(\left\lfloor \frac{M}{N \mod M} \right\rfloor -1\right)(N \mod M) \geq (N \mod M).$$

As all M processors have no tardiness in period 0, a processor that is never selected up to period $j+1$ still has no tardiness, thus, the $(N \mod M)$ never selected processors are also available in period $j+1$.

Case 2: $j=\left\lfloor \frac{M}{N \mod M} \right\rfloor -1$.

Consider period $j-q_0$. By the discussion in case 1, there are at least $(N \mod M)$ processors that have never been selected before and still have tardiness equal to zero. Thus, $(N \mod M)$ zero tardiness processors are selected and reach tardiness $\Lambda_0=\lambda$ in period $j-q_0+1$. Their cooldown time is q_0 and by Lemma 17 will be available again in period $j+1$. Hence $|A_{j+1}| \geq (N \mod M)$.

Case 3: $j \geq \left\lfloor \frac{M}{N \mod M} \right\rfloor$.

First observe that $j \geq q_{>}=\lfloor L/\mu \rfloor$, by equation (7). Then notice that, by Corollary 4, in each period up to j processors have either increased their tardiness by λ, decreased it by μ, or reset it, and thus hypothesis 2 of Lemma 18 is true, i.e., $\forall m \in [1..M], \exists i,h \geq 0 : R_{m,k}=i\lambda-h\mu$. Then Lemma 18 might be applied in this case for period j. This means that

$$|S_{j,1} \cup S_{j,2} \cup S_{j,3}| \geq N \mod M.$$

As, by Remark 8 the three sets are disjoint, and by Remark 9, the three sets are subsets of A_{j+1}, the thesis follows from

$$|S_{j,1} \cup S_{j,2} \cup S_{j,3}| \leq |A_{j+1}|.$$

□

7 Schedule hyper-period

In this section, we prove the result of the exact hyper-period length for uniform instances anticipated in Theorem 10, that is the number of periods needed for the schedule to return to a previous tardiness configuration. This result derives from the deterministic nature of the scheduler and shows that the same scheduling repeats periodically.

With respect to the conference version of the paper Buzzega et al. (2023), rewrite some results and adapt them to the stronger hypotheses needed for dealing with any work-conserving non-preemptive schedulers.

Theorem 10 (Hyper-period length) *Given a difficult instance* (N, L, M, P) *, let* $u \in \mathbb{N}^+$ *be the maximum number of times a processor is selected before resetting. Then the hyper-period length is equal to* $\left\lceil u\frac{L}{\mu} \right\rceil$.

Proof By Theorem 7 and Corollary 9, any period has a standard configuration start and by Lemma 3 standard configuration applies. By Corollary 4 selected processors increase their tardiness by λ and non-selected ones decrease it by μ, w.r.t the previous period. For each of the u times in which the processor is selected, one period is used to increase the tardiness to Λ_i and q_i periods to reduce it to $((i+1)\lambda \mod \mu)$ for some $i \geq 0$.

After that, the remaining tardiness will be $(u\lambda \mod \mu)$. If it is greater than 0, one more period is needed to reset it. Thus, the number of periods needed to reset is exactly:

$$\begin{aligned}
&\sum_{i=0}^{u-1}(1+q_i) + \left(\left\lceil \frac{u\lambda \mod \mu}{\mu} \right\rceil\right) \\
&= u + \frac{u\lambda + (0\lambda \mod \mu) - ((u)\lambda \mod \mu)}{\mu} + \frac{(u\lambda \mod \mu) + (-u\lambda \mod \mu)}{\mu} \\
&\overset{(6)}{=} \frac{uL + (-uL \mod \mu)}{\mu} \\
&= \left\lceil u\frac{L}{\mu} \right\rceil.
\end{aligned}$$

□

Theorem 11 *For any difficult instance* (N, L, M, P) *,* u^* *is the maximum number of times any processor is selected before resetting.*

Proof We begin by showing that there exists a finite value $u \in \mathbb{N}^+$ that corresponds to the number of times any processor is selected before resetting.

As in the proof of Theorem 10, in every period, the standard configuration applies and any non-zero tardiness may be expressed as a multiple of λ minus a multiple of μ. By Remark 1, a selected processor m must have initial tardiness $\in [0..\mu - 1]$, so for any $j \geq 0$ and $m \in S_j$, $R_{m,j} = (i\lambda \mod \mu)$ for some $i \geq 0$. In particular, there exists a finite value $ord(\lambda)$ called additive order of $\lambda \in \mathbb{Z}_\mu$ such that $ord(\lambda) \cdot \lambda \equiv 0 \mod \mu$. It follows that the number of tardiness increases u does not exceed the order: $u \leq ord(\lambda)$, thus u exists and is finite.

Now we prove that $u^* = u$.

Consider any processor m with zero tardiness in period $j \geq 0$ and that increases its tardiness exactly u times (that is in u different periods) before resetting. Since there are always enough available processors, by Remark 6 m has position lower or equal to $(N \mod M)$ if and only if it is selected. By Theorem 10, m is subject to $\left\lceil \frac{uL}{\mu} \right\rceil$ periods in which it has non-zero tardiness.

By Lemma 14 processor m increases its position u times and, in order to bring it back to a value lower than $(N \mod M)$, it reduces it $\left\lfloor \frac{uM - u(N \mod M)}{N \mod M} \right\rfloor$ times. Thus we must have that

$$\left\lceil \frac{uL}{\mu} s \right\rceil \leq u + \left\lfloor \frac{uM - u(N \mod M)}{N \mod M} \right\rfloor, \tag{18}$$

otherwise m would have position lower or equal to $(N \mod M)$, causing it to be selected $u+1$ times without resetting, which contradicts the hypothesis. Since by definition, u^* is the minimum value for which this inequality holds, we have that $u^* \leq u$.

On the other hand, we supposed that m does not reset its tardiness before having been selected u times, hence, using Corollary 4, the number of periods needed to be selected $u' < u$ times and lower the tardiness to a value in $[0..\mu]$ is at least as much as the periods needed to reduce the position to a value lower or equal to $(N \mod M)$:

$$\left\lceil \frac{u'L}{\mu} \right\rceil - 1 \geq \left\lfloor \frac{u'M}{N \mod M} \right\rfloor, \quad \forall 0 < u' < u.$$

This implies that u is the smallest value for which Eq. (18) holds; hence $u = u^*$. □

For the following corollary, the proof can be found in the conference version of the paper.

Corollary 19 (Buzzega et al. 2023) *We have that* $u^* \leq \min\left(\frac{\mu}{\gcd(L,\mu)}, \frac{N \mod M}{\gcd(N,M)}\right)$, *hence it can be computed in time* $O(\min(M, L))$ *that is pseudo-polynomial in the length of the input*.

8 Conclusions and future work

In this paper, we complete the study of the tardiness of uniform instances, introduced in our previous work (Buzzega et al. 2023), which are a subset of Harmonic Task systems. We consider tasks to be synchronous and periodic, with the period and job length to be shared among the tasks. We present a full analytical study of the tardiness of such instances under the general class of non-preemptive and work-conserving schedulers.

We show that the job length is a tight bound to tardiness and we provide a method to find the exact scheduling tardiness as well as the length of the hyperperiod, both in pseudo-polynomial time, linear with the number of processors. In particular, given a uniform instance (N, L, M, P), to derive the results, one has to first compute the following values:

- value λ as $\lceil N/M \rceil L - P$, in constant time;
- value μ as $P - \lfloor N/M \rfloor L$, in constant time;
- value u^* as the minimum positive integer u such that $\lceil uL/\mu \rceil \leq uM/(N \mod M)$, in time linear with $\min(M, L)$, because u^* is bounded by the minimum between $\mu/gcd(L, \mu)$ and $(N \mod M)/gcd(N, M)$.

Then, it is possible to compute:

- the exact instance tardiness as the maximum between zero and $\lambda + \max_{0 \le i < u^*}(i\lambda \mod \mu)$, in time linear with u^* (i.e., as before, in time linear with $\min(M, L)$);
- the exact length of the hyper-period as $\left\lceil \frac{u^* L}{\mu} \right\rceil$, in constant time.

We observe that each value can be computed with simple algebraic operations in a number that depends on M, the number of processors. Since in practical applications, the latter is generally relatively small, the method is computationally tractable for real-world scenarios in which uniform instances arise.

We performed some tests to support the latter affirmation: we computed values of u^* for various instances by forcing $u^* > 1$, otherwise the computation requires only less than ten arithmetical and logic operations. At this aim, we chose M and P of the same order of magnitude, uniformly up to a value of 4 billion (far larger than typical applications), and N was sampled to be up to 16 times larger than M. Value L was set so that $L \ge \left\lfloor \frac{P}{\lfloor N/M \rfloor + 1/\lfloor M/(N \mod M) \rfloor} \right\rfloor$, so that $u^* > 1$, with values never smaller than $M/16$; i.e., $\min(M, L) \in \Omega(M)$. Among the 1 million tests we conducted, only 0.04% of the instances had u^* that exceeded the value of 5000, and for all other instances, u^* was computed in less than 0.46 ms. In general, the execution time never exceeded 95 ms. All experiments were carried out on a Python 3.12.5 implementation of the algorithm, run on an Apple MacBook with an M2 chip. Since the computation of u^* is to be done only once to determine the instance tardiness, tests confirm tractability for real-world scenarios.

As for future works, we plan to use the gained insights to explore the behavior of more general task systems. By systematically relaxing constraints, we aim to deepen our understanding of scheduling problems and generalize our results. The aim of such research direction is twofold: on the theoretical side, to analytically derive tighter bounds to tardiness, while on the practical side, to overcome a limitation of the present study and include more general types of instances that are more applicable to real-world scenarios. This iterative process of generalization will guide our future research efforts in this area.

Acknowledgements The authors thank Gianluca Nocetti for the insightful discussions that led to a preliminary version of this paper.

Funding Open access funding provided by Università di Pisa within the CRUI-CARE Agreement. Giovanni Buzzega is supportd, in part, by MUR of Italy, under PRIN Project n. 2022TS4Y3N—EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data.

References

Ahmed S, Anderson JH (2021) Tight tardiness bounds for pseudo-harmonic tasks under global-edf-like schedulers. In: 33rd Euromicro conference on real-time systems, ECRTS 2021, July 5–9, 2021, Virtual Conference. https://doi.org/10.4230/LIPIcs.ECRTS.2021.11

Anssi S, Kuntz S, Gérard S, Terrier F (2013) On the gap between schedulability tests and an automotive task model. J Syst Architect 59(6):341–350. https://doi.org/10.1016/j.sysarc.2013.02.001

Buzzega G, Nocetti G, Montangero M (2023) Characterizing G-EDF scheduling tardiness with uniform instances on multiprocessors. In: ACM Proceedings of the 31st international conference on real-time networks and systems, RTNS 2023, Dortmund, Germany, June 7–8, 2023, pp 45–55. https://doi.org/10.1145/3575757.3593641

Chen C, Mohan S, Pellizzoni R, Bobba RB, Kiyavash N (2019) A novel side-channel in real-time schedulers. In: 25th IEEE real-time and embedded technology and applications symposium, RTAS 2019, Montreal, QC, Canada, April 16–18, 2019, pp 90–102. https://doi.org/10.1109/RTAS.2019.00016

Devi UC, Anderson JH (2008) Tardiness bounds under global EDF scheduling on a multiprocessor. Real Time Syst 38(2):133–189. https://doi.org/10.1007/s11241-007-9042-1

Dong Z, Yang K, Fisher N, Liu C (2021) Tardiness bounds for sporadic gang tasks under preemptive global EDF scheduling. IEEE Trans Parallel Distrib Syst 32(12):2867–2879. https://doi.org/10.1109/TPDS.2021.3081019

Erickson J (2014) Tardiness bounds and overload in soft real-time systems. PhD thesis, University of North Carolina, Chapel Hill, USA. https://doi.org/10.17615/fvp3-q039

Erickson JP, Devi U, Baruah SK (2010a) Improved tardiness bounds for global EDF. In: 22nd Euromicro conference on real-time systems, ECRTS 2010, Brussels, Belgium, July 6–9, 2010, pp 14–23. https://doi.org/10.1109/ECRTS.2010.25

Erickson J, Guan N, Baruah SK (2010b) Tardiness bounds for global EDF with deadlines different from periods. In: Principles of distributed systems—14th international conference, OPODIS 2010, Tozeur, Tunisia, December 14–17, 2010. Proceedings, pp 286–301. https://doi.org/10.1007/978-3-642-17653-1_22

Erickson JP, Anderson JH, Ward BC (2014) Fair lateness scheduling: reducing maximum lateness in g-edf-like scheduling. Real Time Syst 50(1):5–47. https://doi.org/10.1007/s11241-013-9190-4

Fu Y, Kottenstette N, Chen Y, Lu C, Koutsoukos XD, Wang H (2010) Feedback thermal control for real-time systems. In: 16th IEEE real-time and embedded technology and applications symposium, RTAS 2010, Stockholm, Sweden, April 12–15, 2010, pp 111–120. https://doi.org/10.1109/RTAS.2010.9

Goh J, Anderson JH (2023) Reducing response-time bounds via global fixed preemption point edf-like scheduling. In: 29th IEEE international conference on embedded and real-time computing systems and applications, RTCSA 2023, Niigata, Japan, August 30–Sept. 1, 2023, pp 117–126. https://doi.org/10.1109/RTCSA58653.2023.00023

Kato S, Ishikawa Y, Rajkumar R (2011) Cpu scheduling and memory management for interactive real-time applications. Real-Time Syst 47(5):454–488. https://doi.org/10.1007/s11241-011-9127-8

Leoncini M, Montangero M, Valente P (2017) A branch-and-bound algorithm to compute a tighter bound to tardiness for preemptive global EDF scheduler. In: ACM Proceedings of the 25th international conference on real-time networks and systems, RTNS 2017, Grenoble, France, October 04–06, 2017, pp 128–137. https://doi.org/10.1145/3139258.3139277

Leoncini M, Montangero M, Valente P (2019) A parallel branch-and-bound algorithm to compute a tighter tardiness bound for preemptive global EDF. Real Time Syst 55(2):349–386. https://doi.org/10.1007/s11241-018-9319-6

Leontyev H, Anderson JH (2007) Tardiness bounds for FIFO scheduling on multiprocessors. In: 19th Euromicro conference on real-time systems, ECRTS'07, 4–6 July 2007, Pisa, Italy, Proceedings, p 71. https://doi.org/10.1109/ECRTS.2007.33

Leontyev H, Anderson JH (2008) A unified hard/soft real-time schedulability test for global EDF multiprocessor scheduling. In: Proceedings of the 29th IEEE real-time systems symposium, RTSS

2008, Barcelona, Spain, 30 November–3 December 2008, pp 375–384. https://doi.org/10.1109/RTSS.2008.15

Leontyev H, Anderson JH (2009) Generalized tardiness bounds for global multiprocessor scheduling. Real-Time Syst 44(1–3):26–71. https://doi.org/10.1007/s11241-009-9089-2

Li H, Sweeney J, Ramamritham K, Grupen R, Shenoy P (2003) Real-time support for mobile robotics. In: The 9th IEEE real-time and embedded technology and applications symposium, 2003. Proceedings, pp 10–18. https://doi.org/10.1109/RTTAS.2003.1203032

Mills AF, Anderson JH (2010) A stochastic framework for multiprocessor soft real-time scheduling. In: 2010 16th IEEE real-time and embedded technology and applications symposium. https://doi.org/10.1109/rtas.2010.33

Mills AF, Anderson JH (2011) A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In: 2011 IEEE 17th international conference on embedded and real-time computing systems and applications. https://doi.org/10.1109/rtcsa.2011.30

Shih C, Gopalakrishnan S, Ganti P, Caccamo M, Sha L (2003) Scheduling real-time dwells using tasks with synthetic periods. In: Proceedings of the 24th IEEE real-time systems symposium (RTSS 2003), 3–5 December 2003, Cancun, Mexico, pp 210–219. https://doi.org/10.1109/REAL.2003.1253268

Valente P (2016) Using a lag-balance property to tighten tardiness bounds for global EDF. Real Time Syst 52(4):486–561. https://doi.org/10.1007/s11241-015-9237-9

Voronov S, Anderson JH, Yang K (2018) Tardiness bounds for fixed-priority global scheduling without intra-task precedence constraints. In: Proceedings of the 26th international conference on real-time networks and systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10–12, 2018, pp 8–18. https://doi.org/10.1145/3273905.3273913

Yun H, Ali W, Gondi S, Biswas S (2017) Bwlock: a dynamic memory access control framework for soft real-time applications on multicore platforms. IEEE Trans Comput 66(7):1247–1252. https://doi.org/10.1109/tc.2016.2640961

Giovanni Buzzega is a Ph.D. Student at the Dipartimento di Informatica of the University of Pisa (Italy). He obtained his Bachelor's and Master's degrees in Computer Science from the University of Modena and Reggio Emilia in 2020 and 2022, respectively. In 2023, he was a research fellow at the University of Modena and Reggio Emilia. His research interests include algorithms on strings and graphs in the context of bioinformatics as well as real-time scheduling.

Manuela Montangero is an Associate Professor at the Dipartimento di Scienze Fisiche, Informatiche e Matematiche of the University of Modena and Reggio Emilia (Italy). Her research interests include algorithms and distributed algorithms, social and multimedia computing, scheduling analysis, and combinatorics. She got her Laurea cum Laude in Computer Science from the University of Pisa in 1996. In 2001, she got a Ph.D. in Computer Science from the University of Salerno. From 2001 to 2004, she was at the Istituto di Informatica e Telematica of the National Research Council (CNR) of Pisa, with a temporary research associate position. Since 2005, she has been at the University of Modena and Reggio Emilia.