Noname manuscript No.
(will be inserted by the editor)

Using a Lag-Balance Property to Tighten Tardiness Bounds for Global EDF

Paolo Valente

the date of receipt and acceptance should be inserted later

Abstract Several tardiness bounds for global EDF and global-EDF-like schedulers have been proposed over the last decade. These bounds contain a component that is explicitly or implicitly proportional to how much the system may be cumulatively lagging behind, in serving tasks, with respect to an ideal schedule. This cumulative lag is in its turn upper-bounded by upper-bounding each per-task component *in isolation*, and then summing individual per-task bounds. Unfortunately, this approach leads to an over-pessimistic cumulative upper bound. In fact, it does not take into account a *lag-balance* property of any work-conserving scheduling algorithm.

In this paper we show how to get a new tardiness bound for global EDF by integrating this property with the approach used to prove the first tardiness bounds proposed in the literature. In particular, we compute a new tardiness bound for implicit-deadline tasks, scheduled by preemptive global EDF on a symmetric multiprocessor. According to our experiments, as the number of processors increases, this new tardiness bound becomes tighter and tighter than the tightest bound available in the literature, with a maximum tightness improvement of 29%. A negative characteristic of this new bound is that computing its value takes an exponential time with a brute-force algorithm (no faster exact or approximate algorithm is available yet).

As a more general result, the property highlighted in this paper might help to improve the analysis for other scheduling algorithms, possibly on different systems and with other types of task sets. In this respect, our experimental results also point out the following negative fact: existing tardiness bounds for global EDF, including the new bound we propose, may become remarkably loose if every task has a low *utilization* (ratio between the execution time and the minimum inter-arrival time of the jobs of the task), or if the sum of the utilizations of the tasks is lower than the total capacity of the system.

Paolo Valente
Department of Physics, Computer Science and Mathematics
Via Campi 213/A - 41125 Modena, Italy
E-mail: paolo.valente@unimore.it

1 Introduction

Many time-sensitive applications have soft real-time requirements, i.e., tolerate deadline misses, provided that some appropriate service-quality requirement is met. Small-scale examples range from infotainment to non-safety-critical control systems, while large-scale examples range from financial to IPTV services. In many cases, a sufficient service-quality requirement is that an application-specific, maximum *tardiness* is guaranteed with respect to deadlines (Kenna et al (2011)). Meeting this requirement may even allow all deadlines to be met, where buffers can be used to compensate for fluctuations of job completion times.

Guaranteeing deadlines, or at least a bounded tardiness, to time-sensitive applications is complicated by the fact that the growing computational demand of these applications can now be met only using multiprocessors. Optimal multiprocessor scheduling algorithms, guaranteeing all deadlines to feasible task sets, have been devised by Anderson and Srinivasan (2004); Baruah et al (1996); Megel et al (2010); Regnier et al (2011). These algorithms are relatively complex, as guaranteeing all deadlines on a multiprocessor is not a trivial task. Sub-optimal but simpler algorithms are available too (Devi and Anderson (2005); Valente and Lipari (2005b); Erickson and Anderson (2012); Erickson et al (2014)). One such algorithm is Global Earliest Deadline First (G-EDF). G-EDF has been proved to guarantee a bounded tardiness to feasible task sets by Devi and Anderson (2005); Valente and Lipari (2005b). From empirical results, Bastoni et al (2010) have also inferred that G-EDF is an effective solution for soft real-time (SRT) tasks on (clusters of) up to 8 processors. Variants of G-EDF have been defined as well (Erickson and Anderson (2012); Erickson et al (2014)). In these variants, named G-EDF-like (GEL) schedulers, each job is assigned a fixed priority as in G-EDF, but, differently from G-EDF, this priority is not a function of only the deadline of the job. In more detail, the priority of each job is computed so as to optimize further goals in addition to guaranteeing a bounded tardiness.

Both G-EDF and GEL schedulers, besides being simpler than optimal scheduling algorithms, cause a lower overhead than the latter (apart from possible GEL schedulers with job priorities computed through complex formulas or algorithms). Finally, differently from optimal algorithms, G-EDF and GEL schedulers enjoy the *job-level static-priority* (JLSP) property, which is required in most synchronization solutions (Brandenburg (2011)).

On the downside, being that G-EDF and GEL schedulers are not optimal, they may or may not succeed in meeting the time requirements of an application, depending on the maximum tardiness that they can guarantee to the tasks of the application. Maximum tardiness also determines buffer sizes where buffers are used to conceal deadline misses, and memory may be a critical resource in, e.g., embedded systems. As a consequence, *tightness* is an important property of tardiness bounds for these schedulers. In this respect, in this paper we compute a new tardiness bound for G-EDF and implicit-deadline tasks. This bound proved to be tighter than existing ones within the scope of our experiments. Additionally, although we focus only on the just-mentioned pair of scheduling algorithm and task model, we leverage a property that holds for any work-conserving scheduler, as discussed in the description of our contribution.

Related work

Initial tardiness bounds for G-EDF were computed by comparing G-EDF against an ideal schedule (Devi and Anderson (2005); Valente and Lipari (2005b)).[1] Devi and Anderson (2005) also proposed, without proof, an improvement over the original bound reported in the paper. The same authors subsequently proved this improvement in (Devi and Anderson (2008)).

A new technique for computing worst-case tardiness bounds for G-EDF was then proposed by Erickson et al (2010). Using this technique, named *compliant-vector analysis* (CVA), the authors obtained tardiness bounds that dominate the one proved in (Devi and Anderson (2005)). They also obtained bounds that dominate the one computed in (Devi and Anderson (2008)), by combining CVA with the same improvement proved in (Devi and Anderson (2008)).

Erickson and Anderson (2012) then presented an alternative improvement over CVA, which they named *PP Uniform Reduction Law*. Finally, Erickson et al (2014) turned the computation of the smallest-possible tardiness bounds with CVA and the PP Uniform Reduction Law into a linear program. More precisely, the last two papers focused on job *lateness*, defined as just the difference between completion time and deadline, but the provided results can of course be used also to compute bounds to the tardiness (defined, instead, as the maximum between 0 and the lateness).

To discuss the tightness of the bounds described so far, we can consider the following common, best-case definition: a tardiness bound for a scheduling algorithm S is *tight*, for a number of processors M, if there exists at least one task set such that, if the task set is scheduled with the algorithm S on M processors, then the maximum tardiness experienced by at least one task of the set is equal to the value of the bound for that task. Unfortunately, except for the case $M = 2$ (Devi and Anderson (2008)), existing tardiness bounds for G-EDF have not been proved to be tight, even with respect to this best-case definition. This leaves room for the existence of tighter bounds, which is exactly what we show, experimentally, in this paper.

Contribution

In this paper we show how to get a new tardiness bound for G-EDF, by integrating the approach used by Devi and Anderson (2008) with a *balance* property that G-EDF shares with any work-conserving scheduling algorithm. In particular, we compute such a bound for the following case: (1) *implicit-deadline sporadic* tasks scheduled by *preemptive* G-EDF, (2) a system made of M identical, unit-speed processors, and (3) no synchronization among tasks.

To compute this new bound, we use the same approach as in (Devi and Anderson (2005, 2008); Valente and Lipari (2005b)). As in these works, we compute a bound

[1] Unfortunately, the proofs in (Valente and Lipari (2005b)) contain an error. As shown in an amended, but not peer-reviewed version of the paper (Valente and Lipari (2005a)), if the part of the proofs containing that error is removed, then the rest of the proofs still allow a tardiness bound to be proved. But the latter bound is larger than both the problematic bound in (Valente and Lipari (2005b)) and the bound in (Devi and Anderson (2005)).

made of two components. The first component accounts for the fact that, after a new job is released, G-EDF may fail to fully utilize all the processors while executing the jobs scheduled before the new job and the new job itself. This may lower the number of jobs per second executed in parallel, i.e., the total speed of the system. In particular, the system may become slow enough, in executing jobs, to finish the new job later than its deadline.

In addition, exactly because of the above issue, there may be several late jobs at some point in time. Suppose that a new unlucky job arrives at such a time instant. Some of the late jobs may have a higher priority (earlier deadline) than this new job. The new job may then suffer from a tardiness caused not only by the above sub-optimal job-packing issue, but also by the need to wait for the completion of several late jobs before being executed. The second component accounts for the latter additional waiting.[2]

The first component is quite easy to compute, whereas most of the paper is devoted to computing the second component. To this purpose, we turn the last qualitative consideration into the following quantitative relations. First, given the extra work that the system has to do to complete late jobs, we prove (trivially) that the time that the new job has to wait before being executed increases with this extra work. Then, we prove that this extra work grows, in its turn, with how much the system is *cumulatively lagging*, when the new job is released, behind an *ideal schedule*, or equivalently an *ideal system*, in which every job is completed by its deadline. In the end, the second component is proportional to this cumulative lag. According to this fact, we obtain the second component by computing an upper bound to this cumulative lag (more precisely to the cumulative lag for a special subset of tasks, as we explain in Section 4.5). The peculiarity of the proofs reported in this paper lies in how we compute an upper bound to the cumulative lag.

To highlight this peculiarity, we need to add the following piece of information: the cumulative lag is defined as the sum of individual per-task *lags*, where each per-task lag measures how much the system is lagging behind the ideal system in executing the jobs of the task. We get an upper bound to the cumulative lag by the same two steps as in Devi and Anderson (2005, 2008): first, we compute an upper bound to each per-task lag, and then we sum these per-task upper bounds. But, differently from Devi and Anderson (2005, 2008), we do not compute the upper bounds to per-task lags in isolation from each other. In contrast, we prove and use the following *lag-balance* property: for each task, the bound to its lag contains a negative component proportional to the sum of part of the other contributing lags. This sort of internal *counterbalancing* brings down the value of the whole sum. Finally, as for a comparison against CVA analysis (Erickson et al (2010); Erickson and Anderson (2012); Erickson et al (2014)), in CVA analysis the components of a special vector

[2] Actually, the situation is a little bit more complex, because: 1) some processor may become available to execute the new job even before all higher-priority late jobs have been completed, and 2) the execution of the new job may happen to be suspended and restarted several times (as it may happen to any job). But, as we show formally through the lemmas proved in this paper (and as it has been already proved in the literature), the essence of the problem is the same: the more remaining work the system has to do, to complete higher-priority late jobs, the more time the new job may have to wait before being executed.

play a similar role as the lags contributing to Σ. But no balance property is used to upper-bound the sum of these components.

We tag the whole tardiness bound as *harmonic*, because of the relation, highlighted in Section 3, between the dominant term in the bound and a harmonic number. We could have computed an even tighter bound for task sets with a total utilization strictly lower than the system capacity, but this would have made formulas more complicated and proofs longer (see the comment after Lemma 10 for details). Finally, the value of the bound apparently remains the same regardless of whether the improvement proved in (Devi and Anderson (2008)) is applied, i.e., whether it is assumed that one of the lags contributing to the cumulative lag is not higher than the maximum job length. We do not investigate this issue further in this paper.

We evaluated, experimentally, the tightness of the harmonic bound and of the other bounds available in the literature. To this purpose, we simulated the execution of a large number of random task sets. According to our results with up to 8 processors, the bound obtained by combining CVA with the improvement proved in (Devi and Anderson (2008)), which we name CVA2, is the tightest one available in the literature. On the opposite end, the bound computed in (Devi and Anderson (2008)), named DA DA in our results, quickly becomes substantially looser than CVA2 as the number of processors M increases, until it becomes up to about 40% looser than CVA2 with $M = 8$. Nevertheless, the harmonic bound, obtained by integrating the lag-balance property with the approach used to compute DA, is always at least as tight as CVA2, and becomes tighter and tighter than CVA2 as M increases. In particular, the harmonic bound is from 18% to 29% tighter than CVA2 with $M = 8$.

On the opposite end, a negative characteristic of the harmonic bound is that its formula is quite complex, and a brute-force algorithm takes an exponential time to compute the second component of the bound (fortunately, this component has to be computed once per task set, as it is the same for all the tasks in a task set). The constants in the formula of the cost of the algorithm are however so small that computing the harmonic bound was feasible for almost every group of 1000 task sets considered in our experiments (Section 9). The only exceptions were some of the cases where using a tardiness bound is apparently not necessary (discussed below). In addition, the fact that in this paper we show only an exponential algorithm does not imply that polynomial-time, exact or approximate algorithms cannot be devised. Investigating these algorithms is out of the scope of this paper.

As a general result, the lag-balance property described in this paper might help to reduce pessimism on worst-case response-times also with other scheduling algorithms, systems and types of task sets. In fact, the property is so general that one has the impression that it might be possible to extend the harmonic bound to non-preemptive G-EDF or stochastic task-set models, since in both cases existing bounds are already computed using the same approach as the harmonic bound itself (Devi and Anderson (2008); Mills and Anderson (2010)). It would also be interesting to compare the harmonic bound with bounds for improved versions of G-EDF, such as G-FL ((Erickson and Anderson (2012); Erickson et al (2014)).

Finally, our experiments also provide the following general, negative information: all the bounds considered in the experiments, including the harmonic bound, become quite loose when all tasks have a low utilization, or when the total utilization of the

task set is lower than the total capacity of the system. Fortunately, as we discuss in detail in Section 9, in the first case tardiness bounds are apparently not very relevant, whereas the second case is exactly the one for which there is room for improvement for the harmonic bound.

Organization of this paper

In Section 2 we describe the system model. Then we report the harmonic bound in Section 3. In Section 4 we provide the minimal set of definitions needed to give a detailed outline of the proof of the bound. We provide such an outline in Section 5. Although the lag-balance property is the key property that allows us to compute the harmonic bound, the proof of the lag-balance property does not highlight the general characteristics of G-EDF and a multiprocessor system that enable us to prove this property. For this reason we devote Section 6 to the intuition behind this property, or, more precisely, behind the *time-balance* property, which is the preliminary property from which we derive the lag-balance property. The core of the paper then follows: after proving a set of preliminary lemmas in Section 7, we report the proof in Section 8. Finally we report our experimental results in Section 9.

2 Task and service model

In this section we introduce the basic notations used in the paper. All notations are also summarized in Table 1. To justify an equality or an inequality, we often write the reason why that relation holds on the equality/inequality sign. In most cases, we write just the number of the equation or lemma by which that relation holds (see, e.g., the first and last equalities in (2)). As for time intervals, we write $[t_1,t_2]$, $[t_1,t_2)$ or (t_1,t_2), to refer to all the times t such that $t_1 \leq t \leq t_2$, $t_1 \leq t < t_2$ or $t_1 < t < t_2$. For brevity, we often use the notation

$$f(t_1,t_2) \equiv f(t_2) - f(t_1), \tag{1}$$

where $f(t)$ is a any function of the time and any ordering between t_1 and t_2 may hold, i.e., t_1 may be either smaller than or larger than t_2. From this definition, it follows that

$$\begin{aligned}\forall t_1,t_2,t_3 \quad f(t_1,t_2) \stackrel{(1)}{=} -f(t_1)+f(t_2) = -f(t_1)+f(t_3)-f(t_3)+f(t_2) \stackrel{(1)}{=} \\ f(t_1,t_3)+f(t_3,t_2),\end{aligned} \tag{2}$$

regardless of the ordering among the three time instants. Similarly:

$$\forall t_1,t_2 \quad f(t_2) = f(t_2)-f(t_1)+f(t_1) \stackrel{(1)}{=} f(t_1)+f(t_1,t_2). \tag{3}$$

To simplify the notation, in summations over set of tasks we use the symbol of the set of tasks to denote, instead of the set of tasks, the set of indexes of the tasks in the set. Finally, to avoid special considerations for corner cases, we assume that, for any expression x and any pair of integers n_1 and n_2, $\sum_{i=n_1}^{n_2} x = 0$ if $n_1 > n_2$ (note that x may or may not be a function of the index i). In the following two subsections we describe the task and service models.

Table 1: Notations.

τ, N	Set and number of tasks
$\tau_i \in \tau$	i-th task in τ
C_i	Worst-case computation time of the jobs of task τ_i
T_i	Period/Minimum inter-arrival time of the jobs of task τ_i
$U_i = \frac{C_i}{T_i} \leq 1$	Utilization of task τ_i
$U_{sum} = \sum_{i \in \tau} U_i \leq M$	Total utilization of the task set
J_i^j	j-th job of task τ_i
r_i^j, $d_i^j = r_i^j + T_i$	Release time and absolute deadline of the j-th job of task τ_i
f_i^j	Completion time of the j-th job of task τ_i in the MPS
MPS	Real system, made of M unit-speed processors
M	Number of processors of the MPS
DPS	Ideal, reference system, made of one processor per task; the speed of each processor is equal to U_i
$W_i^{MPS}(t)$ / $W_i^{DPS}(t)$	Amount of service (Subsection 4.3) given to task τ_i by the MPS/DPS up to time t
l_i^j	Length of the j-th job of task τ_i
$L_i = \max_j l_i^j$	Maximum length of the jobs of task τ_i; numerically equal to C_i, but measured in service units
$\mathrm{lag}_h\langle d\rangle(t) = W_h^{DPS}(t) - W_h^{MPS-B}\langle d\rangle(t)$	Lag of task τ_h at time t, with $W_h^{MPS-B}\langle d\rangle(t)$ defined by (21)
$\mathrm{lag}_h(t) = \mathrm{lag}_h\langle d_i^j\rangle(t)$	Short form for $\mathrm{lag}_h\langle d_i^j\rangle(t)$
$\tau^{MPS}(\hat{J},t) \subseteq \tau$	*Blocking tasks*, i.e., subset of tasks owning, at time t, pending jobs with a deadline earlier than or equal to that of $\hat{J}$ in the MPS
$\tau^{DPS}(\hat{J},t) \subseteq \tau$	Subset of tasks generated by the algorithm in Definition 6
$\tau(\hat{J},t) \subseteq \tau$	*Extended set of blocking tasks*, defined as $\tau^{MPS}(\hat{J},t) \cup \tau^{DPS}(\hat{J},t)$
$[b_k, f_k)$	k-th non-growing-Lag interval
$\Lambda(k) = \max_{1 \leq p \leq k} \lambda(p)$	Maximum total lag up to time b_k, with the function $\lambda(p)$ defined by (24)
$\hat{\tau}$	Subset of tasks in $\tau(\hat{J}, b_k)$ with a positive lag at time b_k
G	Cardinality of $\hat{\tau}$

2.1 Task model

We consider a set τ of N tasks τ_1, τ_2, ..., τ_N, with each task τ_i consisting of an infinite sequence of jobs J_i^1, J_i^2, ... to execute. The j-th job J_i^j of τ_i is characterized by: a release (arrival) time r_i^j, a computation time c_i^j, equal to the time needed to execute the job on a unit-speed processor, a finish time f_i^j, an absolute deadline d_i^j, within which the job should be finished, and a *tardiness* defined as $\max(0, f_i^j - d_i^j)$.

Each task τ_i is in its turn characterized by a pair (C_i, T_i), where $C_i \equiv \max_j c_i^j$ and T_i is the *minimum inter-arrival time* of the jobs of the task, i.e., $\forall j\ r_i^{j+1} \geq r_i^j + T_i$. No offset is specified for the release time of the first job of any task. There is no synchronization between tasks. The deadline of any job J_i^j is *implicit*, i.e., equal to $r_i^j + T_i$. We say that a job is *pending* during $[r_i^j, f_i^j]$. Note that, according to this definition, a job is pending also while it is being executed.

We define as *tardiness of a task* the maximum tardiness experienced by any of its jobs. Finally, for each task τ_i we define its *utilization* as $U_i \equiv \frac{C_i}{T_i} \leq 1$. We denote by U_{sum} the total utilization $\sum_{i \in \tau} U_i$ of the task set (note that, according to what was

previously said, in the summation we use the symbol τ to denote the set of the indexes of the tasks in τ).

Figure 1.A shows a possible sequence of job arrivals for a four-task set with total utilization 3. The tasks are characterized by the following pairs (C_i, T_i): $(3,4)$, $(3,4)$, $(2,3)$ and $(5,6)$. Every job is represented as a rectangle, with an up-arrow on top of its left side: the projection onto the x-axis of the left extreme of the rectangle represents the release time of the job, whereas the length of the base of the rectangle is the time needed to complete the job on a unit-speed processor. Finally, every job is followed by a down arrow, representing its deadline. The first job of every task is released at time 0, except for J_3^1. The job J_2^3 is the only non-first job that is released later than the deadline of the previous job of the same task.

2.2 Service model

We consider a symmetric multiprocessor comprised of M identical, unit-speed processors, and name this system *MPS* (MultiProcessor System). We assume that $M < N$, that $U_{sum} \leq M$ and that jobs are scheduled according to the global and preemptive EDF (G-EDF) policy: 1) each time a processor becomes idle, the pending (and not yet executing) job with the earliest deadline is dispatched on it, 2) if a job with an earlier deadline than some of the jobs in execution arrives when all the processors are busy, then the job with the latest deadline among those in execution is *preempted*, i.e., it is suspended, and the newly arrived job is started. Ties are arbitrarily broken. From this policy, the following property immediately follows.

Lemma 1 *In the MPS, at all times the jobs in execution have the earliest deadlines among all pending jobs. Ties are arbitrarily broken.*

We say that a task is *being served* while one of its jobs is being executed. In general, the execution of a job may be interrupted several times to grant the processor to other jobs with earlier deadlines. For any job J_i^j, we use the term *portion* as a short form to refer to any of the maximal parts of the job that are executed, without interruption, during $[r_i^j, f_i^j]$. We call *start time* of a portion the time instant at which the portion starts to be executed, and use the notation $\hat{J} \subseteq J_i^j$ to mean that $\hat{J}$ is a portion of a job J_i^j. The portion $\hat{J}$ is pending from time r_i^j to its completion. For brevity, hereafter we say *deadline of* $\hat{J}$ to refer to the deadline d_i^j of the job the portion belongs to. As can be deduced from this notation, we assume that $\hat{J}$ may also be the only portion of J_i^j, which happens if, once started, J_i^j is executed without interruption until it is finished. Accordingly, for brevity, in the rest of the paper we use the term portion to refer both to a proper contiguous slice of a job and to a whole job. To avoid ambiguity, we stress that with the term *job* we always refer to a whole job.

We say that a pending job portion is *blocked* if it cannot start to be executed. In our model a job portion can be blocked for only one of the following two reasons.

1. The first portion of a job J_i^j cannot be started before the preceding job J_i^{j-1} is finished. In this respect, we say that a pending job portion $\hat{J} \subseteq J_i^j$ is *blocked by*

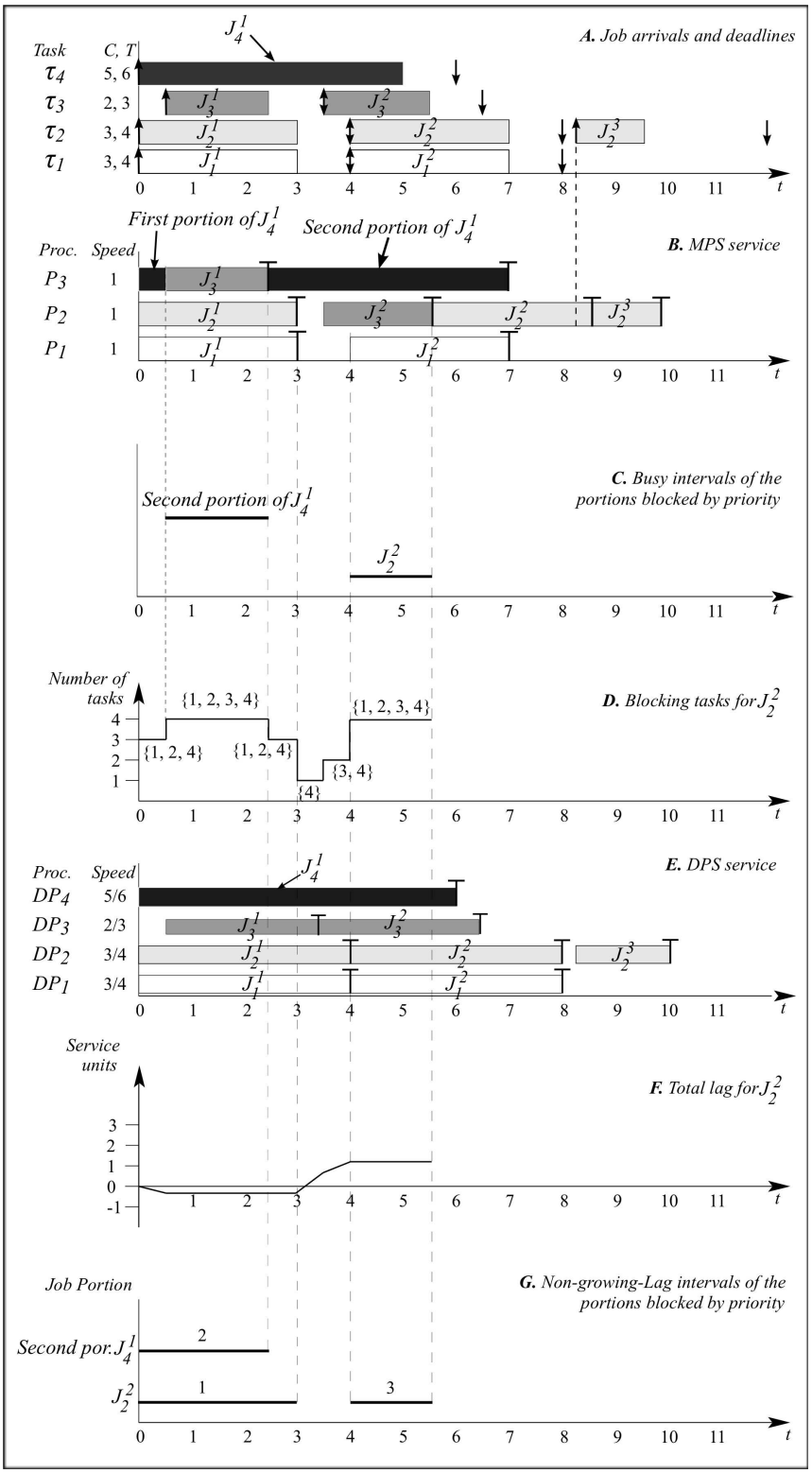


Fig. 1: Example of task set, MPS service, busy intervals, blocking tasks, DPS service, total lag and non-growing-Lag intervals.

precedence at time t if a previous portion of J_i^j or any portion of J_i^{j-1} is still pending at time t.

2. If a pending portion $\hat{f} \subseteq J_i^j$ is not blocked by precedence at a given time t, then the only other reason why it could not be started at that time is because all processors are busy serving portions of jobs with deadlines earlier than or equal to d_i^j. In this case, we say that the portion is *blocked by priority* at time t.

For the sake of clarity, we emphasize that only portions not blocked by precedence may be deemed blocked by priority. Finally, we often use the following phrase for brevity: we say just that a portion is blocked by priority, without specifying the time instant at which, or the time interval during which the portion is in such a state, to mean that the portion happens to be blocked by priority (at least) right before it starts to be executed.

Before discussing the above definitions further, we show them through Figure 1.B, which shows the execution of the jobs depicted in Figure 1.A, on an MPS with three unit-speed processors. For each processor, each rectangle represents the execution of a job portion on that processor, with the projection onto the x-axis of the left/right side of each rectangle representing the start/finish time of the portion. Finally, each (whole) job completion time is highlighted with a $\top$. As an example of job portions, when J_3^1 arrives at time 0.5, J_4^1 is suspended. The second and last portion of J_4^1 is then started at time 2.5 after being blocked by priority during $[0.5, 2.5)$, and it is finished at time 7. The only other portion blocked by priority is the whole job J_2^2, while J_2^3 is an example of job portion blocked by precedence.

Finally, given a maximal time interval during which a task τ_i is continuously served, we define as *chain* (of portions of jobs) of task τ_i, the sequence of portions of jobs of τ_i executed during the time interval. An example of a chain, in Figure 1.B, consists of J_2^2 and J_2^3, executed back-to-back during $[5.5, 10]$. As a degenerate case, a chain may be made of only one portion. In this respect, recall also that we use the term portion as a short form to refer to both an actual portion and a whole job. In addition, a job that starts to be executed right after being blocked by precedence, necessarily belongs to a chain, as it starts to be executed right after the previous job of the same task. Instead, we define as *head* of a chain the job portion executed for first in the sequence. We conclude this section with the following note.

Note 1 *If a chain contains at least two job portions, then the head of the chain is necessarily the last portion of a job. In fact, by definition of job portions, the two job portions cannot belong to the same job, and thus a new job of the same task starts to be executed right after the head.*

3 The harmonic bound

In this section, we report, first, the harmonic bound, and then a brute-force algorithm to compute it. The formula of the harmonic bound is relatively complex. Then, to help the reader better understand the formula and get an idea of the order of magnitude of the bound, we also instantiate the formula for two extreme cases in which it becomes

simpler. To the same purpose, we show, through a numerical example, how to compute the bound with the brute-force algorithm. Finally, we provide a brief analysis of the computational cost of the algorithm, plus some example of the execution time of an implementation of the algorithm.

To write the harmonic bound, we start by introducing the following definition and symbols. The motivation for introducing the following symbols is only ease of notation, and none of them represents, alone, a relevant quantity (they are just components of relevant quantities). The only exception is Ω, for which we describe the quantity it represents when commenting on the harmonic bound.

Definition 1 For any set of tasks $\tau' \subseteq \tau$, containing G tasks, and for all $g \in \{1,2,\ldots,G\}$, let g' denote the index of the g-th task in τ'.

Using this notation, we can define the terms that appear in the formula of the bound (in the next equation, v' is the index of the v-th task in τ' according to Definition 1):

$$\begin{gathered}\forall \tau' \text{ such that } \tau' \subseteq \tau \text{ and } |\tau'| \leq \lceil U_{sum} \rceil - 1, \\ \forall g \leq |\tau'| + 1,\ M_g^{\tau'} \equiv M - \sum_{v=1}^{g-1} U_{v'},\end{gathered} \tag{4}$$

$$\Gamma \equiv M \cdot \max_{\substack{\tau' \subseteq \tau \text{ and} \\ |\tau'| = \lceil U_{sum} \rceil - 1}} \sum_{g=1}^{|\tau'|} \frac{C_{g'}}{M_g^{\tau'}}, \tag{5}$$

and

$$\Omega \equiv \frac{1}{M} \cdot \max_{\substack{\tau' \subseteq \tau \text{ and} \\ |\tau'| \leq \lceil U_{sum} \rceil - 1}} \left[M_{|\tau'|+1}^{\tau'} \left(\Gamma \sum_{g=1}^{|\tau'|} \frac{U_{g'}}{M_g^{\tau'} M_{g+1}^{\tau'}} + \sum_{g=1}^{|\tau'|} \frac{C_{g'}}{M_g^{\tau'}} \right) \right]. \tag{6}$$

We can now state the harmonic bound. In particular, first we report the bound, and then we show that its value is positive. After that, we explain where the components of the bound stem from. Finally, we analyze the bound in two extreme cases, and show why we name it *harmonic*.

Theorem 1 (Harmonic Bound) *For every job J_i^j,*

$$f_i^j - d_i^j \leq \Omega + \frac{M-1}{M} C_i. \tag{7}$$

As stated in the following lemma, the right-hand side (RHS) of (7) is not negative. Therefore it is not necessary to take the maximum between its value and 0 to turn it into a tardiness bound (recall that tardiness is a non-negative quantity).

Lemma 2 *The RHS of* (7) *is not negative, and is equal to* 0 *if and only if* $M = 1$.

Proof First, since in both (5) and (6),

$$|\tau'| \leq \lceil U_{sum} - 1 \rceil \leq M - 1, \tag{8}$$

we have

$$\forall g \leq |\tau'|+1 \quad M_g^{\tau'} = M - \sum_{v=1}^{g-1} U_{v'} \geq M - \sum_{v=1}^{(|\tau'|+1)-1} U_{v'} \overset{(8)}{\geq} M - \sum_{v=1}^{[(M-1)+1]-1} U_{v'} \overset{U_{v'} \leq 1}{\geq} M - \sum_{v=1}^{M-1} 1 = M - (M-1) = 1 > 0. \tag{9}$$

The above inequality, plus $U_{v'} > 0$ and $C_{v'} > 0$, imply that all the terms in (5) and (6), including the sums, are non-negative. In particular, if $M = 1$, then all the sums are null. In fact, in this case $|\tau'| = 0$ in both (5) and (6) by (8). In the end, $\Gamma \geq 0$ and $\Omega \geq 0$, and, if $M = 1$ then $\Gamma = \Omega = 0$. As for the term $\frac{M-1}{M} C_i$, its value is non-negative because $M - 1 \geq 0$ and $C_i > 0$. In particular, $\frac{M-1}{M} C_i = 0$ if and only if $M = 1$. □

In the introduction we said that the tardiness bound reported in this paper contains two components. The term $\frac{M-1}{M} C_i$ in the RHS of (7) is the first component mentioned in the introduction. That is, this term accounts for the maximum tardiness that can be caused by the fact that the MPS is sub-optimal in packing jobs on processors, and hence in utilizing the full speed of the system. For example, the MPS fails in utilizing all the processors while the job J_4^1 is pending in Figure 1.B, and (the second portion of) J_4^1 ends up being completed after its deadline. By properly breaking jobs into smaller portions and scheduling these portions in a more clever way, it would have been possible to fully utilize all processors and complete J_4^1 by its deadline (Anderson and Srinivasan (2004)).

Instead, the quantity Ω is the second component of the bound we mentioned in the introduction. As we already stated, this component stems from the following issue: exactly because of the above sub-optimal job-packing problem, the MPS may be already late in executing other higher-priority jobs when a new job arrives. By Lemma 1, the MPS may have to complete part of these late jobs before one processor becomes available to execute the new job. The component Ω accounts for the additional delay caused by this fact. In particular, as we also already said, given an ideal system that completes every job by its deadline, this additional delay is proportional to how much the MPS may be cumulatively lagging behind the ideal system, in terms of amount of service received by tasks, when a new job arrives. Therefore, we obtain Ω by computing an upper bound to this cumulative lag.

As a final consideration on Ω, we can note that Ω has the same value for every task. This may pave the way to even tighter bounds. In fact, for example, the bound computed by Erickson et al (2010) contains, for each task, a component equivalent to Ω, but whose value varies (also) with the parameters of the task. That bound is tighter than previous ones exactly because of this characteristic.

Finally, we consider two extreme cases to get an idea of the order of magnitude of the bound (7). First, $U_{g'} \to 0$ for all τ' and for all $g \leq |\tau'|$. This implies $M_g^{\tau'} \to M$. Defining $C \equiv \max_{\tau_i \in \tau} C_i$, and considering also that $|\tau'| \leq M - 1$ by (8), we get,

from (7),

$$f_i^j - d_i^j \leq \frac{1}{M}\left[M(0+C\sum_{g=1}^{M-1}\frac{1}{M})\right] + \frac{M-1}{M}C_i = \frac{M-1}{M}C + \frac{M-1}{M}C_i = \frac{M-1}{M}(C+C_i). \tag{10}$$

On the opposite side, if $U_{g'} \to 1$ for all τ' and for all $g \leq |\tau'|$, then $M_g^{\tau'} \to M-(g-1)$. Assuming also that τ' is as large as possible, i.e., that $|\tau'| = M-1$, the sums in (5) and (6) can be loosely upper-bounded as follows

$$\sum_{g=1}^{|\tau'|}\frac{C_{g'}}{M_g^{\tau'}} \leq C\sum_{g=1}^{M-1}\frac{1}{M_g^{\tau'}} \overset{U_{g'}\to 1}{\longrightarrow} C\sum_{g=1}^{M-1}\frac{1}{M-(g-1)} = C\sum_{g=2}^{M}\frac{1}{g}, \tag{11}$$

$$\sum_{g=1}^{|\tau'|}\frac{U_{g'}}{M_g^{\tau'}M_{g+1}^{\tau'}} \overset{\substack{U_{g'}\to 1,\\ |\tau'|=M-1}}{\longrightarrow} \sum_{g=1}^{M-1}\frac{1}{(M-(g-1))(M-g)} = \sum_{g=2}^{M}\frac{1}{g(g-1)} < 1, \tag{12}$$

where the last inequality follows from the well-known equality $\sum_{g=1}^{\infty} 1/(g\cdot(g+1)) = 1$. In addition, if $U_{g'} \to 1$ for all τ' and for all $g \leq |\tau'|$, then $M_{|\tau'|+1}^{\tau'} \to M-(|\tau'|+1-1) = M-(M-1+1-1) = 1$. Replacing (11), (12) and $M_{|\tau'|+1}^{\tau'} \to 1$ in (7), we get

$$\begin{aligned} f_i^j - d_i^j \leq \frac{1}{M}\left[\left(M\cdot C\sum_{g=2}^{M}\frac{1}{g}\right) + \sum_{g=2}^{M}\frac{1}{g}\right] + \frac{M-1}{M}C_i = \\ C\left(\sum_{g=2}^{M}\frac{1}{g}\right)\left(1+\frac{1}{M}\right) + \frac{M-1}{M}C_i. \end{aligned} \tag{13}$$

As M increases, the RHS of (13) quickly becomes higher than the RHS of (10), because of the term $C\sum_{g=2}^{M}\frac{1}{g}$. In particular, as M increases, this term becomes the dominant one in the RHS of (13). In this respect, the sum $\sum_{g=2}^{M}\frac{1}{g}$ is equal to the harmonic number H_M minus 1, while the whole term $C\left(\sum_{g=2}^{M}\frac{1}{g}\right)$ is an upper bound to the quantity to which the sum $\sum_{g=1}^{|\tau'|}\frac{C_{g'}}{M_g^{\tau'}}$ tends, in both (5) and (6), as $U_{g'} \to 1$. We name the bound (7) *harmonic bound* after these facts.

It is easy to show that the RHS of (13) grows less than linearly with M (a coarse upper bound to the order of growth of the RHS of (13) is $\log M$). Similarly, one can see numerically that the RHS of (7), and in particular, its component Ω, grows less than linearly with M as well. The small slope of the RHS of (7) is the key property for which the harmonic bound proves to be tighter than existing ones in our experiments. As we highlight in Section 9.1, such a small slope follows from the fact that, differently from the literature, we use the lag-balance property when computing an upper bound to the cumulative lag of the MPS with respect to an ideal system.

3.1 A brute-force algorithm for computing the harmonic bound

The hardest part in computing the value of the harmonic bound is of course computing Γ and Ω. A brute-force algorithm for computing these values is the following. First, compute Γ. To this purpose, generate, as a first step, all the possible subsets $\tau' \subseteq \tau$ of $\lceil U_{sum} \rceil - 1$ tasks. Then, for every subset τ' generated, compute the value of the sum in (5) for all the possible orderings of the tasks in τ' (i.e., all the $|\tau'|$-tuples containing all the tasks in τ'). In fact, for a given subset τ' and for every $g \leq |\tau'|$, the value of the sum in (5) may change depending on which task is considered to be the g-th task in τ' (this point should become clearer from the example that we give in a moment). As a final step, use the maximum value of the sum obtained from the previous steps and multiply it by M to get Γ. Once computed Γ, use the same approach to compute Ω, taking into account that, this time, all the subsets of *at most* $\lceil U_{sum} \rceil - 1$ tasks need to be generated (and not only all the subsets of exactly $\lceil U_{sum} \rceil - 1$ tasks).

This algorithm can be refined in such a way to reduce, when possible, the number of subsets to generate. To introduce this improvement, we can consider that, given two subsets of tasks τ'_A and τ'_B, it is not necessary that $\tau'_A = \tau'_B$ holds for the two subsets to yield the same value for the *max* functions in (5) and (6). In fact, a sufficient condition is that $|\tau'_A| = |\tau'_B|$, and, for every $g = 1, 2, \ldots, |\tau'_A|$, both the g-th task in τ'_A and the g-th task in τ'_B are characterized by the same pair (C_i, T_i). As a consequence, to compute the *max* functions in (5) and (6) it is enough to restrict the search (and thus the generation of subsets) to any maximal set of subsets such that, for each element of the set, there is no other element for which the above condition holds.

We show now an example of the steps taken by this brute-force algorithm on a real task set (more in general, this provides also a practical example of how the harmonic bound can be computed). Consider a task set made of four tasks, with each task characterized by the following pairs (C_i, T_i): $\tau_1 \to (4,5)$, $\tau_2 \to (4,5)$, $\tau_3 \to (4,5)$ and $\tau_4 \to (3,5)$. Accordingly, $U_{sum} = 3$ and hence $\lceil U_{sum} \rceil - 1 = 2$. All the possible sets τ' in (5) consist therefore of two tasks, and the argument of the max function in (5), which we denote by $Arg^{\Gamma}(\tau_g, \tau_h)$, can be rewritten as

$$Arg^{\Gamma}(\tau_g, \tau_h) = \frac{C_g}{3} + \frac{C_h}{3 - U_g}. \tag{14}$$

The whole formula can then be rewritten as

$$\Gamma = 3 \cdot \max_{\substack{\tau_g \in \tau, \\ \tau_h \in \tau, \\ g \neq h}} \left[Arg^{\Gamma}(\tau_g, \tau_h) \right]. \tag{15}$$

According to the above considerations on how to reduce the number of subsets to generate, to compute the value of the RHS of (15) it is enough to consider any possible maximal set of pairs of tasks (τ_g, τ_h), such that each pair in the set may yield a different value for the argument of the *max* function in (15).[3] Since τ_1, τ_2 and τ_3 have

[3] To compute (15), we have to consider pairs of tasks (τ_g, τ_h), and not just sets $\{\tau_g, \tau_h\}$, because the value of $Arg^{\Gamma}(\tau_g, \tau_h)$ may change if the order of the argument changes, i.e., $Arg^{\Gamma}(\tau_g, \tau_h) \neq Arg^{\Gamma}(\tau_h, \tau_g)$ may hold.

the same parameters, one such possible maximal set is: $\{(\tau_1,\tau_2),(\tau_1,\tau_4),(\tau_4,\tau_1)\}$. Using this maximal set, we get, from (15),

$$\begin{aligned}\Gamma &= 3\cdot\max\left(\frac{C_1}{3}+\frac{C_2}{3-U_1},\frac{C_1}{3}+\frac{C_4}{3-U_1},\frac{C_4}{3}+\frac{C_1}{3-U_4}\right)= \\ &3\cdot\max\left(\frac{4}{3}+\frac{4}{3-4/5},\frac{4}{3}+\frac{3}{3-4/5},\frac{3}{3}+\frac{4}{3-3/5}\right)= \\ &3\cdot\max\left(\frac{104}{33},\frac{85}{33},\frac{8}{3}\right)=3\cdot\frac{104}{33}=\frac{104}{11}=9.45.\end{aligned} \tag{16}$$

The next step is computing Ω. According to (6) and to the fact that $\lceil U_{sum}\rceil-1=2$, the sets τ' to consider may consist of one or two tasks. If τ' contains just one task, i.e., if $\tau'=\{\tau_g\}$ for some $\tau_g\in\tau$, then the argument of the max function in (6) is a function of τ_g. Denoting this argument by $Arg^{\Omega}(\tau_g)$, from (6) we get

$$Arg^{\Omega}(\tau_g)=(3-U_g)\left(\Gamma\frac{U_g}{3\cdot(3-U_g)}+\frac{C_g}{3}\right). \tag{17}$$

In a similar vein, if τ' contains two tasks, i.e., $\tau'=\{\tau_g,\tau_h\}$, then the argument of the max function in (6), say $Arg^{\Omega}(\tau_g,\tau_h)$, becomes

$$Arg^{\Omega}(\tau_g,\tau_h)=(3-U_g-U_h)\left[\Gamma\left(\frac{U_g}{3\cdot(3-U_g)}+\frac{U_h}{(3-U_g)\cdot(3-U_g-U_h)}\right)+\frac{C_g}{3}+\frac{C_h}{3-U_g}\right]. \tag{18}$$

Accordingly, we can rewrite (6) as follows:

$$\Omega=\frac{1}{3}\cdot\max\left\{\max_{\tau_g\in\tau}\left[Arg^{\Omega}(\tau_g)\right],\max_{\substack{\tau_g\in\tau,\\ \tau_h\in\tau,\\ g\neq h}}\left[Arg^{\Omega}(\tau_g,\tau_h)\right]\right\}. \tag{19}$$

Similarly to the steps taken to compute Γ from (15), we can compute the value of the first argument of the external max function in (19) by considering any maximal set of tasks such that the value of $Arg^{\Omega}(\tau_g)$ may vary over the elements of the set. A possible maximal set with this property is $\{\tau_1,\tau_4\}$. Instead, regarding $Arg^{\Omega}(\tau_g,\tau_h)$, a possible maximal set of pairs of tasks (τ_g,τ_h) such that the value of $Arg^{\Omega}(\tau_g,\tau_h)$ may differ over the elements of the set is: $\{(\tau_1,\tau_2),(\tau_1,\tau_4),(\tau_4,\tau_1)\}$. Expanding the RHS of (19) according to these two sets, i.e., with the same procedure as that by which we obtained (16) from (15), and computing the value of the resulting expression, we get $\Omega=3.15$. Replacing this value in (7), we get the following tardiness upper bound for every job of τ_1, τ_2 or τ_3: 5.82, and the following bound for every job of τ_4: 5.15.

3.2 Computational cost of the brute-force algorithm

We evaluate now the worst-case computational cost of the brute-force algorithm defined in Section 3.1. To this purpose, we suppose that all the possible subsets τ' of $U_{sum}-1$ tasks have to be considered to compute Γ in (5), and that all the possible subsets τ' of at most $U_{sum}-1$ tasks have then to be considered to compute Ω in (6).

In such a case, the order of the number of subsets to generate, and therefore the order of the cost of the brute-force algorithm as a function of N, is

$$\binom{N}{\lceil U_{sum} \rceil - 1} \leq \binom{N}{N/2} < \frac{2^N}{\sqrt{\frac{N}{2}\pi}}, \tag{20}$$

where the second inequality is one of the well-known simple upper bounds to the central binomial coefficient (Koshy (2008)).

Finally, as an example of the actual execution time of the brute-force algorithm, we provide such an example through the C implementation we made for the experiments (Experiment-scripts (2014)). To compute the bound for all the tasks in a worst-case task set, this C implementation takes, on an Intel Core i72760QM, from about $23\mu s$ if $M = 2$ and $N = 3$, to about $17s$ if $M = 8$ and $N = 17$. Proposing also more efficient algorithms to compute or approximate the harmonic bound is out of the scope of this paper.

4 Preliminary definitions

Unfortunately, our proof of the harmonic bound is rather long. To help the reader not to get lost in such a long proof, we break it as much as possible into manageable pieces. To this purpose, we start by introducing, in this preliminary section, only the definitions needed to provide a detailed outline of the proof itself. The major part of these definitions basically coincide with corresponding definitions in (Devi and Anderson (2008); Valente and Lipari (2005b)). Some concepts are however expressed in a slightly different way, to better comply with the proof strategy adopted in this paper.

Most of the following time intervals and quantities are defined as a function of a generic portion $\hat{J}$, belonging to a job J_i^j. These are the portion and the job that we use as a reference, in the proof, to prove (7).

4.1 Busy interval and blocking tasks

The following time interval basically coincides with that defined in (Devi and Anderson (2008)[Definition 4]).

Definition 2 Given a job portion $\hat{J} \subseteq J_i^j$ blocked by priority and starting at time $\hat{s}$, we define as *busy interval* for the portion $\hat{J}$, the maximal time interval, starting not before time r_i^j and ending at time $\hat{s}$, during which all processors are constantly busy executing jobs with a deadline earlier than or equal to that of $\hat{J}$.

For example, as shown in Figure 1.C, the busy interval for the second portion of J_4^1 is $[0.5, 2.5)$. The busy interval for a job portion $\hat{J}$ is a superset of the time interval during which the portion is blocked by priority. In fact, by definition, while $\hat{J}$ is blocked by priority, all processors are constantly busy executing jobs with a deadline

earlier than or equal to that of $\hat{J}$. But the same may happen also while $\hat{J}$ is blocked by precedence.

To be able to block the execution of $\hat{J}$, a task must own at least one pending job with a deadline earlier than or equal to d_i^j. This holds also for τ_i itself, if the portion $\hat{J}$ is blocked by precedence. In fact, the jobs of τ_i that precede J_i^j have an earlier deadline than d_i^j. To give a concise name to all the tasks that can cause a portion $\hat{J}$ to be blocked, we provide the following definition.

Definition 3 We define as *set of blocking tasks* $\tau^{MPS}(\hat{J},t) \subseteq \tau$ for the portion $\hat{J} \subseteq J_i^j$ at time t, the set of tasks owning pending jobs[4], in the MPS and at time t, with deadlines earlier than or equal to d_i^j.

We add the superscript MPS in $\tau^{MPS}(\hat{J},t)$ to distinguish this set from the *extended* set of blocking tasks that we define in Section 4.5. We stress that, by Definition 3, the set $\tau^{MPS}(\hat{J},t)$ may include τ_i itself. To help visualize this set, Figure 1.D shows both the cardinality of $\tau^{MPS}(J_2^2,t)$ and the contents of the set (through the labels on the curve). Every job that is pending during $[0,5.5)$ happens to have a deadline earlier than or equal to $d_2^2 = 8$. Therefore $\tau^{MPS}(J_2^2,t)$ consists, at every time instant in $[0,5.5)$, of the set of all the tasks with pending jobs.

Differently from the busy interval, the set of blocking tasks for a portion $\hat{J} \subseteq J_i^j$ is intentionally well-defined also *before* time r_i^j. We discuss the reason behind this choice in Section 4.5.

4.2 Service unit, processor speed and amount of service

Our proof of the harmonic bound contains several lemmas about amounts of service received by tasks, or about differences between amounts of service. To measure the amount of service received by a task, we assume, first, that any job portion is made of a number of *service units* to execute, equal to the number of time units needed to execute the portion on a unit-speed processor (as the processors of the MPS). We define this number as *length* of the portion. Accordingly, we define the length l_i^j of a job J_i^j as the sum of the lengths of its portions, and, for the overall task τ_i, we define $L_i \equiv \max_j l_i^j$. Of course, numerically, $l_i^j = c_i^j$ and hence $L_i = C_i$ (we use different symbols for lengths because the latter are measured in service units and not in time units). Finally, we (re)define the *speed of a processor* as the number of service units per time unit that the processor completes while executing a job.

As a next step to define a measure of the service received by a task τ_i in a system S, we denote by $I_i^1(t)$, $I_i^2(t)$, ..., $I_i^V(t)$ the maximal time intervals, in $[0,t]$, such that, during each of these time intervals, τ_i is continuously served on the same processor of the system S. Finally, we denote by $I_i(t)$ the union of these time intervals. For example, according to Figure 1.B, in the MPS we have $I_4(8) = \{[0,0.5],[2.5,7]\}$ and $I_3(5) = \{[0.5,2.5],[3.5,5]\}$. For every time interval $I_i^v(t)$ with $v = 1,2,\ldots,V$, we define as *amount of service* received by τ_i during $I_i^v(t)$, the number of service units

[4] Recall that, by our definition of *pending*, also jobs in execution are still pending.

executed for τ_i during $I_i^y(t)$. This quantity is in its turn equal to the length of the time interval $I_i^y(t)$, multiplied by the speed of the processor on which the jobs of τ_i are executed during $I_i^y(t)$. Using this definition, we define the amount of service $W_i^S(t)$ received by a task τ_i in a system S during $[0,t]$, as the sum of the amounts of service received by the task during the time intervals $I_i^1(t)$, $I_i^2(t)$, ..., $I_i^V(t)$.

For the MPS, we denote the last quantity by $W_i^{MPS}(t)$. Since all the processors of the MPS have unit speed, $W_i^{MPS}(t)$ is equal, numerically, to the total time spent by the processors of the MPS executing the jobs of τ_i during $[0,t]$ (but the unit of measure differ). For example, considering Figure 1.B, $W_1^{MPS}(4) = 3$, $W_3^{MPS}(4) = 2.5$ and $W_4^{MPS}(4) = 2$.

4.3 Dedicated-Processor System (DPS)

As we discussed in the introduction and in Section 3, we compute the component Ω in (7) by computing an upper bound to how much the MPS may be cumulatively lagging behind an ideal system when a new job arrives. In this subsection we introduce the ideal system by which we compute this cumulative lag. Such a system is equivalent to the PS schedule used, e.g., in (Devi and Anderson (2005, 2008))[5].

Definition 4 Given a task set τ, we define as *Dedicated-Processor System* (DPS), an ideal system containing, for each task in τ, a *dedicated processor* that executes only the jobs of that task, at a constant speed equal to the utilization U_i of the task.

The execution time of a job J_i^j on the dedicated processor associated to τ_i is then $\frac{c_i^j}{U_i} \leq T_i$, because $c_i^j \leq C_i$ and $U_i = \frac{C_i}{T_i}$ (Section 2). This property, plus the fact that each processor executes only the jobs of its associated task, guarantee that the DPS completes every job no later than its deadline.

As an example, Figure 1.E shows both the characteristics of the DPS associated to the task set in Figure 1.A, and the execution of the jobs on that DPS. The execution of the jobs is represented with rectangles, whose left and right extremes have the same meaning as for the MPS in Figure 1.B. The height of each rectangle is equal to the speed of the dedicated processor executing the job, with the same scale as in Figure 1.B. In other words, the area of the rectangle representing the execution of a job in the DPS is equal to the sum of the areas of the rectangles representing the execution of the portions of the same job in the MPS. Since, for each task τ_i, every job in Figure 1.A has an execution time equal to C_i, apart from J_2^3, the DPS completes every job exactly on its deadline, apart from J_2^3.

Similarly to $W_i^{MPS}(t)$, we denote by $W_i^{DPS}(t)$ the *amount of service* received by τ_i in the DPS during $[0,t]$. $W_i^{DPS}(t)$ is equal to the total time during which the i-th dedicated processor executes jobs of τ_i in $[0,t]$, multiplied by U_i. For example, in Figure 1.E we have $W_1^{DPS}(4) = 4 * (3/4) = 3 = W_1^{MPS}(4)$, whereas $W_3^{DPS}(4) = 3.5 * (2/3) = 2.\overline{3}$, which is lower than $W_3^{MPS}(4) = 2.5$. On the opposite side, $W_4^{DPS}(4) = 4 * (5/6) = 3.\overline{3}$, which is higher than $W_4^{MPS}(4) = 2$.

[5] A PS schedule is an ideal fluid schedule in which each task executes at a precisely uniform rate given by its utilization. In (Devi and Anderson (2008)) the PS schedule is defined formally in Equation (7).

4.4 Lag of a task

As we already said in the introduction, we define the cumulative lag of the MPS as the sum of per-task lags. We define the latter quantity as follows (the following definition is the counterpart of the lag as defined by Equation (8) in (Devi and Anderson (2008)).

Definition 5 Given a task τ_h, a generic time instant t, a generic deadline d of some job, and the maximum index $m \equiv \max_{d_h^n \leq d} n$, i.e., the index m of the latest-deadline job, of τ_h, among those with a deadline earlier than or equal to d, we define

$$W_h^{MPS-B}\langle d\rangle(t) \equiv \min\left[W_h^{MPS}(t), \sum_{n=1}^{m} l_h^n\right], \tag{21}$$

where l_h^n is the length of the n-th job of the task τ_h, and

$$\mathrm{lag}_h\langle d\rangle(t) \equiv W_h^{DPS}(t) - W_h^{MPS-B}\langle d\rangle(t). \tag{22}$$

To measure how much the MPS is lagging behind the DPS in serving τ_h, one would have probably expected the simpler and more natural difference $\mathrm{lag}_h(t) \equiv W_h^{DPS}(t) - W_h^{MPS}(t)$. Instead, we consider a lower-bounded difference that takes into account the service provided to τ_h by the MPS only with respect to jobs with a deadline at most equal to d. We use this lower-bounded difference because it greatly simplifies several formulas and proof steps. To give some examples, we can continue the example we made at the end of Section 4.3. Using as a reference the deadline $d_2^2 = 8$ of J_2^2, we have $\mathrm{lag}_1[8](4) = 0$, $\mathrm{lag}_3[8](4) = -0.1\overline{6}$ and $\mathrm{lag}_4[8](4) = 1.\overline{3}$.

We already said that we use a generic job J_i^j as a reference in the proof. As a consequence, we use the quantity $\mathrm{lag}_h\langle d_i^j\rangle(t)$ very often. For this reason, for ease of notation, we define the following short form for $\mathrm{lag}_h\langle d_i^j\rangle(t)$:

$$\mathrm{lag}_h(t) \equiv \mathrm{lag}_h\langle d_i^j\rangle(t), \tag{23}$$

and, in the rest of the paper, we always use the short form $\mathrm{lag}_h(t)$ instead of the full expression $\mathrm{lag}_h\langle d_i^j\rangle(t)$. In particular, we refer to the quantity $\mathrm{lag}_h(t)$ also when we use the phrase *lag of τ_h at time t*. Finally note that, though the terms tardiness and lag may be somehow interchangeable in natural language, the concept of tardiness, as defined in Section 2, must not be confused with the concept of lag, as defined in (22) (and measured in service units).

4.5 Total lag

As we already explained in Section 3, we compute Ω by computing an upper bound to the cumulative lag of the MPS with respect to the DPS. We define now precisely this cumulative lag. By Lemma 1, given the last portion $\hat{J}$ of the job J_i^j in (7), the only jobs that determine the delay by which $\hat{J}$ is completed are the jobs with a deadline earlier than or equal to d_i^j. Accordingly, by Definition 3, the natural set of tasks to consider

for computing the cumulative lag that influences the tardiness of J_i^j is $\tau^{MPS}(\hat{J},t)$. Unfortunately, considering just this set of tasks further complicates proofs. Instead, we get simpler proofs if we define the cumulative lag as a function of a more *artificial* superset of $\tau^{MPS}(\hat{J},t)$. This extended set may contain also some tasks that do not belong to $\tau^{MPS}(\hat{J},t)$. To define this extended set, we start from the following intermediate set.

Definition 6 Given a job portion $\hat{J} \subseteq J_i^j$ blocked by priority and a generic time instant t, we define the set of tasks $\tau^{DPS}(\hat{J},t)$ as the set generated by the following algorithm:

Initial state. At system start-up, the set is empty, i.e., $\tau^{DPS}(\hat{J},0) = \emptyset$.

Entrance (insertion) of a task. A task enters $\tau^{DPS}(\hat{J},t)$ when it exits $\tau^{MPS}(\hat{J},t)$, if the lag of the task is non-positive. More formally, if t_e is a generic time instant at which a task τ_h exits $\tau^{MPS}(\hat{J},t_e)$, but such that $\text{lag}_h(t_e) \leq 0$ holds, then $\tau^{DPS}(\hat{J},t_e^+) = \tau^{DPS}(\hat{J},t_e) \cup \{\tau_h\}$, where $\tau^{DPS}(\hat{J},t_e^+)$ is the content of $\tau^{DPS}(\hat{J},t)$ right after time t_e.

Exit (extraction) of a task. A task exits $\tau^{DPS}(\hat{J},t)$ if its lag becomes strictly positive. More formally, if t_e is a generic time instant at which $\text{lag}_h(t)$ becomes strictly positive for a task τ_h in $\tau^{DPS}(\hat{J},t_e)$, then $\tau^{DPS}(\hat{J},t_e^+) = \tau^{DPS}(\hat{J},t_e) \setminus \{\tau_h\}$.

For example, according to Figure 1.D, the task τ_3 is the first task to exit $\tau^{MPS}(J_2^2,t)$. It happens at time 2.5, because the task stops having pending jobs in the MPS. But, according to Figure 1.E, the task has still a pending job in the DPS at time 2.5. Thus, $\text{lag}_3(2.5) \leq 0$ and hence $\tau^{DPS}(J_2^2,2.5^+) = \tau^{DPS}(J_2^2,2.5) \cup \{\tau_3\} = \{\tau_3\}$. For the same reason, $\tau^{DPS}(J_2^2,3^+) = \tau^{DPS}(J_2^2,3) \cup \{\tau_1,\tau_2\} = \{\tau_1,\tau_2,\tau_3\}$. The first task to exit $\tau^{DPS}(J_2^2,t)$ is τ_2, right after time 4, because $\text{lag}_2(4^+) > 0$. Finally, it is pointless to consider the contents of the set $\tau^{DPS}(J_2^2,t)$ after the start time, 5.5, of J_2^2.

Using the set $\tau^{DPS}(\hat{J},t)$, we can define the set of tasks that we use to define the cumulative lag.

Definition 7 We define as *extended set* $\tau(\hat{J},t)$ of blocking tasks for a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, the union $\tau^{MPS}(\hat{J},t) \cup \tau^{DPS}(\hat{J},t)$.

We can now define formally the cumulative lag of the MPS. We use the name *total lag* for the exact quantity that we use in the proof, to distinguish it from the informal quantity mentioned so far.

Definition 8 We define as *total lag* for a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, the sum $\sum_{h \in \tau(\hat{J},t)} \text{lag}_h(t)$.

The above quantity is the counterpart of the total lag in (Devi and Anderson (2008)[Equation (11)]). Recall that, by (23), $\text{lag}_h(t) = \text{lag}_h\langle d_i^j \rangle(t)$ in Definition 8. Figure 1.F shows the graph of the total lag for J_2^2 during $[0,5.5)$. As can be seen by comparing Figure 1.F with figures 1.D, 1.B and 1.E, the total lag for J_2^2 decreases or is at most constant while $|\tau^{MPS}(J_2^2,t)| \geq 3$ holds, i.e., while $|\tau^{MPS}(J_2^2,t)|$ is greater than or equal to the number of processors M. As it will be possible to deduce from the proof of Lemma 16, the reason is that during these time intervals the MPS provides the tasks in $\tau^{MPS}(J_2^2,t)$, and hence in $\tau(J_2^2,t)$, with a total amount of service per time unit equal to M, while the total speed of the DPS can be at most equal to $U_{sum} \leq M$.

In contrast, the total lag happens to grow while $|\tau^{MPS}(J_2^2,t)| \leq 2$, i.e., during $[3,4)$, because the sum of the speeds of the busy dedicated processors in the DPS happens to be higher than 2, i.e., than the sum of the speeds of the busy processors in the MPS.

As $\tau^{MPS}(\hat{J},t)$, also $\tau(\hat{J},t)$ is well-defined also before the release time r_i^j of J_i^j. The purpose of having these sets well-defined also before r_i^j is having the total lag well-defined before r_i^j too. This is instrumental in our approach for computing an upper bound to the total lag, because we compute this upper bound by *following* the evolution of the total lag also during the time interval $[0,r_i^j]$. In particular, denoting by $\hat{s}$ the start time of $\hat{J}$, we compute an upper bound to the total lag by induction on the beginning of every sub-interval of $[0,\hat{s}]$ during which the total lag does not grow. In the next subsection we define the concepts we use to implement this strategy.

4.6 Growing-Lag and non-growing-Lag intervals

To compute an upper bound to the total lag, we consider, separately, the time intervals during which the total lag grows or does not grow, respectively. To give a shorter name to these intervals, we borrow from Devi and Anderson (2008) the capitalized term *Lag* as a synonym of total lag. Using this term, we give the following definition.

Definition 9 We define as *growing-Lag* or *non-growing-Lag* interval for a portion $\hat{J} \subseteq J_i^j$ blocked by priority and starting at time $\hat{s}$, every maximal time interval in $[0,\hat{s})$ such that the total lag for $\hat{J}$ (Definition 8) is, respectively, strictly increasing at all times in the interval or not increasing at any time in the interval.

During non-growing-Lag intervals the total lag may then even decrease. Figure 1.G shows non-growing-Lag intervals and, by difference, growing-Lag intervals for the job portions blocked by priority (for the moment do not consider the numbers above the non-growing-Lag intervals). The graph of the total lag for J_2^2 in Figure 1.F allows non-growing-Lag intervals to be immediately deduced for J_2^2. On the other hand, the non-growing-Lag interval for the second portion of J_4^1, namely $[0,2.5)$, can be deduced by the fact that the deadlines of the jobs released during $[0,2.5)$ are earlier than both d_1^4 and d_2^2. Then, according to Definition 3, during $[0,2.5)$ the sets of blocking task for the second portion of J_4^1 and for J_2^2 coincide. Hence, during $[0,2.5)$ the graph of the total lag for the second portion of J_4^1 coincides with the graph of the total lag for J_2^2 in Figure 1.F.

4.7 Definitions used in the computation of an upper bound to the total lag

As we already stated, one of the main steps of the proof is computing an upper bound to the cumulative lag, i.e., to the total lag introduced in Definition 8. We compute this bound by induction on the start times of non-growing-Lag intervals. To this purpose, we order non-growing-Lag intervals, globally, by start times, as detailed in the following definition.

Definition 10 Consider the union of all the non-growing-Lag intervals for all the portions blocked by priority. Consider then the *global sequence* of the start times of these non-growing-lag intervals, i.e., the union of the sequences of the start times of the non-growing-lag intervals for each portion blocked by priority. Given this global sequence of start times, we define as k-th non-growing-Lag interval, with $k \geq 1$, the non-growing-Lag interval whose start time is the k-th one in the global sequence. If two non-growing-Lag intervals start at the same time, we assume that the tie is broken arbitrarily.

The numbers reported above non-growing-Lag intervals in Figure 1.G are an example of a possible valid ordering. Hereafter we use the symbols b_k and f_k to denote the start and finish times of the generic k-th non-growing-Lag interval $[b_k, f_k)$. Note that we assume non-growing-Lag intervals to be right-open intervals, in accordance to the fact that a busy interval for a portion is a right-open interval, and, as we prove in Lemma 19, terminates at the same time instant as the last non-growing Lag interval for the portion.

To implement our proof by induction of an upper bound to the total lag, we also use a helper quantity $\Lambda(k)$. To define this quantity, we define, firstly, the maximum total lag for the k-th non-growing-Lag interval as follows

$$\lambda(k) \equiv \sum_{h \in \tau(\hat{f}_k, b_k)} \text{lag}_h\langle \hat{d}_k \rangle (b_k), \tag{24}$$

where $\hat{f}_k$ is the job portion for which the total lag at the start time b_k of the k-th non-growing-Lag interval is maximum, among the portions whose last non-growing-Lag interval is $[b_k, f_k)$, and $\hat{d}_k$ is the deadline of $\hat{f}_k$ (i.e., the deadline of the job $\hat{f}_k$ belongs to). In (24) we cannot use the short expression $\text{lag}_h(b_k)$ as in Definition 8, because the job to which $\hat{f}_k$ belongs may not be J_i^j as in Definition 8. As an example of $\lambda(k)$, consider the third non-growing Lag interval in Figure 1.G, i.e., the interval $[b_3, f_3) = [4, 5.5)$. According to Figure 1.B, this interval is the last non-growing Lag interval for only J_2^2. Moreover, from the graph of the total lag for J_2^2 in Figure 1.F and the values of the lags computed in the example right after Definition 5, we have that the total lag for J_2^2 is equal to $1.1\overline{6}$ at time $b_3 = 4$. As a consequence, $\lambda(3) = 1.1\overline{6}$.

Using the function $\lambda(\cdot)$, we define $\Lambda(k)$ as the maximum among the values of $\lambda(\cdot)$ at times $b_1, b_2, \ldots, b_k$, i.e.,

$$\Lambda(k) \equiv \max_{1 \leq p \leq k} \lambda(p). \tag{25}$$

Figure 2 shows an example of how $\lambda(k)$ and $\Lambda(k)$ may vary with k. Note that, since $\Lambda(k)$ is the maximum possible value of the total lag at the beginning of every non-growing-Lag interval starting in $[0, b_k]$, it is easy to show that $\Lambda(k)$ is more in general the maximum possible value of the total lag, for any job portion, at all times in $[0, b_k]$. Therefore, to compute an upper bound to the total lag for any possible job portion, we compute an upper bound to $\Lambda(k)$ that holds for all k.

In particular, we achieve the latter goal by computing an upper bound to $\lambda(k)$ that holds for all k, and, in more detail, we get an upper bound to $\lambda(k)$ by computing an upper bound to $\sum_{h \in \tau(\hat{f}, b_k)} \text{lag}_h(b_k)$ that holds for any job portion $\hat{f} \subseteq J_i^j$ for

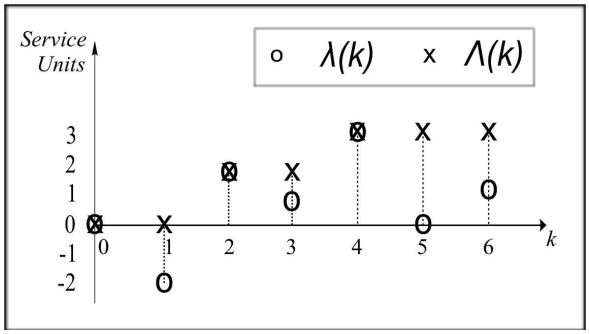


Fig. 2: Possible graphs of $\lambda(k)$ and $\Lambda(k)$: $\lambda(k)$ may fluctuate with k, whereas $\Lambda(k)$ is monotonically non-decreasing. In particular, for all k and for all $p \leq k$, $\Lambda(k) \geq \lambda(p)$.

which the k-th non-growing-Lag interval is the last non-growing-Lag interval (recall that $\text{lag}_h(b_k) = \text{lag}_h\langle d_i^j \rangle(b_k)$ by (23)). A non-trivial aspect related to this step is that $\tau(\hat{J}, b_k)$ may contain up to all the N tasks in τ. Fortunately, exploiting the fact that the sum $\sum_{h \in \tau(\hat{J}, b_k)} \text{lag}_h(b_k)$ is of course at most equal to the sum of only its positive terms, we can upper-bound the sum $\sum_{h \in \tau(\hat{J}, b_k)} \text{lag}_h(b_k)$ by considering far less tasks. In particular, we can consider only the following set of tasks.

Definition 11 Assuming that $\hat{J} \subseteq J_i^j$ is one of the job portions for which $[b_k, f_k]$ is a non-growing-Lag interval, we define $\hat{\tau}$ as the subset of $\tau(\hat{J}, b_k)$ whose tasks have a positive lag at time b_k, i.e.,

$$\hat{\tau} \equiv \{\tau_h | \tau_h \in \tau(\hat{J}, b_k) \wedge \text{lag}_h(b_k) > 0\}. \tag{26}$$

To simplify the notation, we have not included any dependence on k or $\hat{J}$ in the symbol of the set. In fact, in all lemmas we always use the set $\hat{\tau}$ only in relation to the k-th non-growing-Lag interval and the portion $\hat{J}$. From (26), we have

$$\sum_{h \in \tau(\hat{J}, b_k)} \text{lag}_h(b_k) \leq \sum_{g \in \hat{\tau}} \text{lag}_g(b_k). \tag{27}$$

It is then enough to compute an upper bound to the RHS of the last inequality to get an upper bound to the total lag at the beginning of the generic k-th non-growing-Lag interval. This simplifies the proofs, because $\hat{\tau}$ contains at most $\lceil U_{sum} \rceil - 1$ tasks, as stated in Lemma 20.

5 Proof outline

In this section we outline the proof of the harmonic bound, which we provide then in Section 8. In particular, in this section we highlight the main lemmas. The proofs of these lemmas are then provided in Section 8, apart from the next preliminary lemma. Our proof of the harmonic bound moves from the following fact.

Lemma 3 *If a job J_i^j is fully executed without interruption as it is released, then $f_i^j - d_i^j \leq 0$. Otherwise, suppose that J_i^j is still fully executed without interruption, but starts to be executed only (right) after being blocked by precedence. In this case, if $\hat{J}$ is the head of the chain J_i^j belongs to, and $\hat{J} \in J_i^e$, with $e < j$, we have that*

$$f_i^j - d_i^j \leq f_i^e - d_i^e \tag{28}$$

holds, where f_i^e and d_i^e are the completion time and the deadline of J_i^e.

Proof First, $(C_i \leq T_i) \wedge (d_i^j = r_i^j + T_i) \implies r_i^j + C_i \leq d_i^j$. Therefore, if J_i^j is started immediately after its release time r_i^j, and executed without interruption, then $f_i^j = r_i^j + c_i^j \leq r_i^j + C_i \leq d_i^j$. Instead, suppose that the job J_i^j is blocked by precedence and then executed without interruption, as, e.g, J_2^3 in Figure 1.B. In this case, we prove (28) by induction, using the following relations:

$$\begin{aligned} f_i^j - d_i^j = f_i^{j-1} + c_i^j - d_i^j \leq f_i^{j-1} + C_i - d_i^j \leq f_i^{j-1} + T_i - d_i^j = \\ f_i^{j-1} + (r_i^j - r_i^j) + T_i - d_i^j = (f_i^{j-1} - r_i^j) + (r_i^j + T_i) - d_i^j = \\ (f_i^{j-1} - r_i^j) + d_i^j - d_i^j = f_i^{j-1} - r_i^j \leq f_i^{j-1} - d_i^{j-1}, \end{aligned} \tag{29}$$

where the last inequality follows from $r_i^j \geq d_i^{j-1}$. As for a base case, if $j - 1 = e$, then (28) follows directly from (29). As for an inductive step, the inductive hypothesis is that

$$f_i^{j-1} - d_i^{j-1} \leq f_i^e - d_i^e. \tag{30}$$

In this case the thesis follows by substituting (30) in (29). □

From this lemma we can derive the following central corollary for the proof reported in this paper.

Corollary 1 *A necessary condition for a job to experience a non-null tardiness is that either (a) the last portion of the job is blocked by priority, or (b) the whole job is executed after being blocked by precedence, and the head of the chain the job belongs to is in its turn blocked by priority.*

Proof We prove the thesis by contradiction. Suppose that the condition is false. It implies that either the job starts as it is released (because it is not blocked for any reason), or the job is blocked by precedence but the head of the chain it belongs to is a whole job that starts as it is released (by definition of a head, this is the only possibility if the head is not blocked by priority; in fact the head of a chain cannot start right after being blocked only by precedence, because the task is not in service right before the head starts to be executed). In the first case, the tardiness of the job is 0 by Lemma 3, whereas in the second case it is the head to have a null tardiness by Lemma 3. But, by the second part of Lemma 3, the tardiness of the job is therefore 0 also in the second case. □

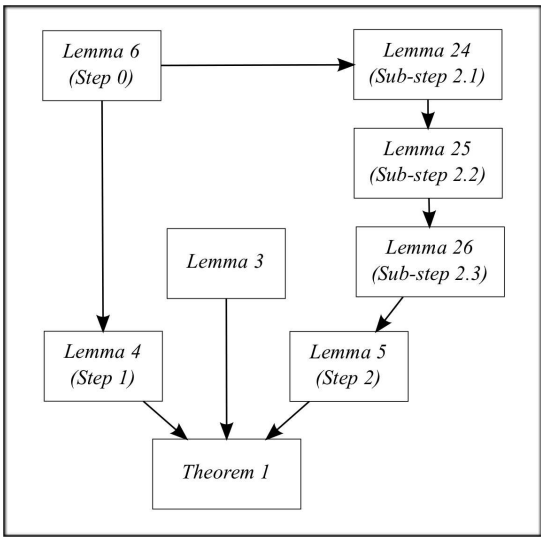


Fig. 3: Dependencies among main lemmas, and between the latter and Theorem 1.

According to Note 1 at the end of Section 2.2, the head of the chain in sub-condition (b) in Corollary 1 is necessarily the last portion of some job. In the end, considering this fact and sub-condition (a) in Corollary 1, to get an upper bound to the tardiness of any job, we need to focus only on *last* portions blocked by priority. This is exactly what we do in most of the proof of the harmonic bound, which consists of the following two main steps. To help the reader visualize the proof flow, we report in Figure 3 the dependencies among main lemmas, and between the latter and Theorem 1 (the figure also shows lemmas and sub-steps that we describe later in Section 8). Finally, the proofs the lemmas reported in the following steps are provided in Section 8.

Step 1: compute an upper bound to the tardiness as a function of the total lag

As we discussed in Section 3, a job J_i^j may experience a positive tardiness, because the service scheme of the MPS is sub-optimal in packing jobs on the processors so as to fully utilize all processors. In particular, exactly because of this sub-optimal job packing, the MPS may be also cumulatively lagging behind in executing some jobs when J_i^j is released. Our first main step is to compute an upper bound to the tardiness as a function of these facts.

Lemma 4 *For every job J_i^j, there exists at least one integer k for which the following inequality holds:*

$$f_i^j - d_i^j \leq \frac{\Lambda(k)}{M} + \frac{M-1}{M} C_i, \tag{31}$$

where, with some abuse of notation, the denominator of the first fraction is measured in service units per time unit[6]*, whereas the coefficient* $\frac{M-1}{M}$ *is a pure number.*

The quantity $\frac{\Lambda(k)}{M}$ is an upper bound to the component of the difference $f_i^j - d_i^j$ caused by the cumulative lag of the MPS, i.e., by the total lag of the MPS with respect to the DPS. Instead, as in the RHS of (7), the *fixed* component $\frac{M-1}{M}C_i$ in the RHS of (31) accounts for the sub-optimal job packing of the MPS.

Step 2: compute an upper bound to the total lag

Whereas the fixed component $\frac{M-1}{M}C_i$ in (31) is already in closed form, the component $\frac{\Lambda(k)}{M}$ is still in open form, as it depends on $\Lambda(k)$. This motivates the second and last main step of the proof: upper-bounding the component $\frac{\Lambda(k)}{M}$ by computing an upper bound to $\Lambda(k)$ that holds for all k. This bound is reported in the next lemma.

Lemma 5 *Let*

$$\Lambda \equiv M \cdot \max_{\substack{\tau' \subseteq \tau \text{ and} \\ |\tau'| = \lceil U_{sum} \rceil - 1}} \sum_{g=1}^{|\tau'|} \frac{L_{g'}}{M_g^{\tau'}}, \tag{32}$$

then we have, for all $k \geq 1$*,*

$$\Lambda(k) \leq \max_{\substack{\tau' \subseteq \tau \text{ and} \\ |\tau'| \leq \lceil U_{sum} \rceil - 1}} \left[M_{|\tau'|+1}^{\tau'} \cdot \left(\Lambda \sum_{g=1}^{|\tau'|} \frac{U_{g'}}{M_g^{\tau'} M_{g+1}^{\tau'}} + \sum_{g=1}^{|\tau'|} \frac{L_{g'}}{M_g^{\tau'}} \right) \right]. \tag{33}$$

We familiarized already with all the terms in (33) in Section 3, except for Λ and $L_{g'}$. In practice, these two terms are the counterparts of Γ and $C_{g'}$ in (5) and (6). In more detail, Λ turns into Γ, and $L_{g'}$ turns into $C_{g'}$ after replacing (33) in (31) (because the denominator in $\frac{\Lambda(k)}{M}$ is a speed, and amounts of services are then turned into times). This replacement is actually our final sub-step for proving Theorem 1.

Regarding the computation of the bound (33), we already said in Section 4.7 that we compute it by computing an upper bound to $\sum_{g \in \tau(\hat{J}, b_k)} \text{lag}_g(b_k)$ that holds for any job portion $\hat{J} \subseteq J_i^j$ for which the k-th non-growing-Lag interval is the last non-growing-Lag interval. In the same section, we already pointed out also that, thanks to (27), we achieve this goal by computing an upper bound to the sum $\sum_{g \in \hat{\tau}} \text{lag}_g(b_k)$.

The peculiarity of the proof reported in this paper lies in *how* we compute an upper bound to the sum $\sum_{g \in \hat{\tau}} \text{lag}_g(b_k)$. In this respect, we compute this upper bound with the same two steps as in (Devi and Anderson (2005, 2008)): first, we compute a bound to each lag in the sum $\sum_{g \in \hat{\tau}} \text{lag}_g(b_k)$, and then we sum these per-task bounds. But we do get a smaller value for the sum of the per-task bounds than that in (Devi and Anderson (2005, 2008)). The reason is that we compute each per-task bound

[6] The denominator of the first fraction is measured in service units per time unit to let the quantity represented by the first fraction be measured in time units. In fact, the other component of the RHS of (31) is measured in time units, whereas $\Lambda(k)$ is measured in service units.

using the lag-balance property, which we can now state more precisely as follows: the upper bound to $\text{lag}_g(b_k)$ for each task $\tau_g \in \hat{\tau}$ decreases as the sum of the lags of part of the other tasks in $\hat{\tau}$ increases. This internal counterbalance enables us to bring down the value of the sum of the per-task bounds over all the tasks in $\hat{\tau}$.

We state the lag-balance property formally in Lemma 24 in Section 8.3 (where we describe the sub-steps by which we implement Step 2). Instead, in the rest of this outline, we focus on the preliminary property from which we derive the lag-balance property. We call this preliminary property *time-balance property*, and we focus on it here, because, for not repeating similar steps twice, we use this property also to compute the bound (31) in Step 1. For this reason, to prove this property is a preliminary Step 0 in our proof.

Step 0: prove the time-balance property

The next lemma states the time-balance property.

Lemma 6 (Time-balance property) *Given: (a) a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, (b) the busy interval $[\hat{b}, \hat{s})$ for $\hat{J}$, (c) any time instant $t_1 \in [\hat{b}, \min\{d_i^j, \hat{s}\})$, and (d) any subset $\Upsilon \subseteq \tau^{MPS}(\hat{J}, t_1)$ not containing τ_i (including the empty set), we have*

$$\hat{s} - d_i^j \leq \frac{\sum_{h \in \tau(\hat{J}, t_1)} lag_h(t_1) - \sum_{v \in \Upsilon} lag_v(\hat{s}) - \hat{l}}{M - \sum_{v \in \Upsilon} U_v}, \tag{34}$$

where $\hat{l}$ is the size, in service units, of the part of the job J_i^j not yet executed in the MPS by time $\hat{s}$, and the quantity $M - \sum_{v \in \Upsilon} U_v$ is measured in service units per time unit (i.e., is a speed).

The above degrees of freedom in choosing t_1 and Υ are key to use the time-balance property for proving the lag-balance property as a part of Step 2. Instead, we use the special case $t_1 = \hat{b}$ and $\Upsilon = \emptyset$ to prove Lemma 4 in Step 1.

Although the time-balance property is the cornerstone on which the harmonic bound depends, the proof of Lemma 6 does not highlight the core properties of G-EDF that enable the time-balance property, and hence the lag-balance property, to hold (the proof of Lemma 6 consists only of algebraic steps). For this reason, we devote the next section to the intuition behind the time-balance property.

6 Intuition behind the time-balance property

The time-balance property, and therefore the lag-balance property, follow from a general balance property that G-EDF shares with any work-conserving scheduler. Hereafter we refer to this property as just *the general property*. In this section we provide, first, a description of this property and an intuitive explanation of the reason why it holds. Then we show, again intuitively, the link between the general property and the time-balance property. The reader not interested in these aspects may skip to the proof machinery in Section 7.

6.1 The general property

Lemma 7 (General property) *Given: (a) a job portion $\hat{J} \subseteq J_i^j$ that starts to be executed at time $\hat{s}$, (b) any time instant t_1, with $t_1 \leq \hat{s}$ and such that at least M tasks have pending jobs at all times in $[t_1, \hat{s})$, (c) any subset $\Upsilon \subseteq \tau$ not containing τ_i (including the empty set), and*

1. *the parameters of every task in τ and the arrival time of every job;*
2. *the amounts of service received by every task, by time t_1, in the MPS and in the DPS, respectively;*
3. *the service pattern of the DPS from time 0 on;*
4. *the amount of service received in the MPS by every task in $\tau \setminus \Upsilon$ during $[t_1, \hat{s})$;*

and, assuming (for simplicity) that for every task $\tau_v \in \Upsilon$ the equality $lag_v(t) = W_v^{DPS}(t) - W_v^{MPS}(t)$ holds at all times t in $[t_1, \hat{s})$, we have that the following implication holds: if all the quantities considered in the above items 1-4 are fixed and if any work-conserving algorithm is used to schedule jobs in the MPS, then the difference $\hat{s} - d_i^j$ decreases as the sum $\sum_{v \in \Upsilon} lag_v(\hat{s})$ increases.[7]

According to the purpose of this section, we discuss and justify Lemma 7 intuitively. First, the general property is more general than the time-balance property because, differently from Lemma 6, in Lemma 7: a) the scheduling algorithm can be any work-conserving algorithm, including, but not limited to, G-EDF, b) t_1 is not constrained to belong to a busy interval, and c) Υ is any subset of tasks that does not contain τ_i. In particular, a task in Υ may or may not be served during $[t_1, \hat{s})$.

To visualize the property, let *Scenario A* and *Scenario B* be two generic scenarios such that:

- for every parameter and quantity in items 1-4 in Lemma 7, the parameter or the quantity has the same value in both scenarios;
- in Scenario A, the value of the sum of the lags $\sum_{v \in \Upsilon} \text{lag}_v(\hat{s})$ is higher than in Scenario B.

The top half of Figure 4 shows the service provided by the MPS and the DPS during $[t_1, \hat{s})$ for a possible Scenario A. The execution of job portions is represented with rectangles. In particular, the job portions of the tasks in/not in Υ are represented with filled/empty rectangles; apart from $\hat{J}$, which is represented with a dark rectangle. Finally, the bottom half of Figure 4 shows the service provided by the MPS and the DPS in a possible Scenario B. Note that, according to item 3 in Lemma 7, the service of the DPS is identical in both scenarios.

As for the service of the MPS in Scenario B, we can consider that each of the parameters and the quantities in items 1-3 in Lemma 7 has the same value in both scenarios. Therefore the only possibility for the sum $\sum_{v \in \Upsilon} \text{lag}_v(\hat{s})$ to be lower in Scenario B (with respect to Scenario A) is that, in Scenario B, the tasks in Υ receive in the MPS more total service, during $[t_1, \hat{s})$, than in Scenario A. Let this extra total service be equal to Δ service units.

[7] For the sake of precision, the invariants considered in this lemma are stronger than those strictly needed for the general property to hold. The reason why we show the property with stronger invariants is that the general property becomes quite hard to visualize otherwise.

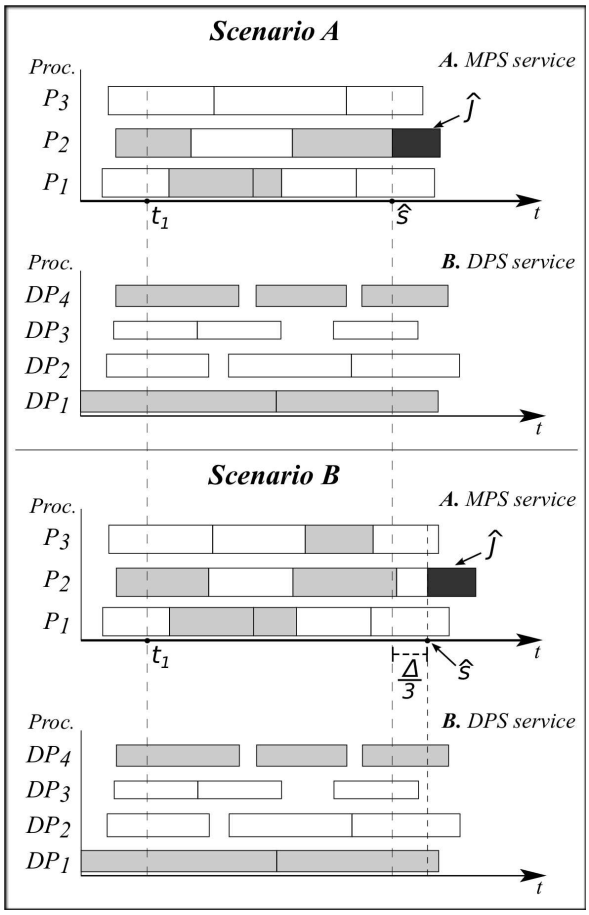


Fig. 4: Example of the general property, with $\Upsilon = \{\tau_1, \tau_4\}$. The execution of job portions of the tasks in/not in Υ is represented by filled yellow/empty rectangles. In scenario B, the tasks in Υ receive Δ extra units of total service in the MPS, during $[t_1, \hat{s})$, with respect to scenario A.

Since, in Scenario B, every task not belonging to Υ receives the same amount of service, in the MPS, as in Scenario A (by item 4), it follows that the overall total work done by the MPS in Scenario B during $[t_1, \hat{s})$ is larger than the total work done by the MPS in Scenario A, exactly by Δ service units. As shown by Figure 4 for the case $M = 3$, this implies that in Scenario B the length of $[t_1, \hat{s})$, i.e., the difference $\hat{s} - t_1$, is larger by Δ/M time units with respect to Scenario A. The reason is that the MPS works at constant total speed M during $[t_1, \hat{s})$, which, in its turn, happens because at least M tasks have pending jobs at all times in $[t_1, \hat{s})$, and the scheduling algorithm is work-conserving.

Before completing the justification of the general property, it may be worth stressing that, being the length of $[t_1, \hat{s})$ in scenario B larger than in scenario A, also the

DPS may happen to provide the tasks in Υ with more total service in Scenario B than in Scenario A. This implies that the difference between the values of $\sum_{v\in\Upsilon} \text{lag}_v(\hat{s})$ in scenario B and scenario A may be less than Δ, whereas one may have expected, in the first place, that it was equal to Δ. The exact relation between Δ and the values of $\sum_{v\in\Upsilon} \text{lag}_v(\hat{s})$ is correctly taken into account (implicitly) in the proof of Lemma 6. In any case, what matters for the general property to hold is that, if the sum of the lags $\sum_{v\in\Upsilon} \text{lag}_v(\hat{s})$ in Scenario B is lower than in Scenario A, then the length of $[t_1,\hat{s})$ is larger too.

To complete the justification of the property, suppose, on the opposite end, that we move from Scenario B to Scenario A, i.e., from an original generic scenario to an alternative scenario in which the value of every parameter and quantity in items 1-4 in Lemma 7 is again the same as in the original scenario, but in which the sum of the lags $\sum_{v\in\Upsilon} \text{lag}_v(\hat{s})$ is higher, and not lower, than in the original scenario. Reversing the above arguments and visualizing what happens through Figure 4, we deduce that now the difference $\hat{s}-t_1$ decreases by Δ/M. Putting this and the previous case together, and considering that, being t_1 fixed, the difference $\hat{s}-d_i^j$ of course grows/decreases with $\hat{s}-t_1$, we get the general property as stated in Lemma 7.

6.2 From the general to the time-balance property

The time-balance property, as stated in Lemma 6, is an instance of the general property in which: 1) the scheduling algorithm is G-EDF, 2) the time interval $[t_1,\hat{s})$ is a subset of the busy period for the job portion $\hat{f}$, and 3) $\Upsilon \subseteq \tau^{MPS}(\hat{f},t_1)$. The second fact is essential for the RHS of (34) in Lemma 6 to contain a component proportional to the total lag. And the presence of this component is the property that allows us to use Lemma 6 for proving Lemma 4 in Step 1. On the other hand, the relation $\Upsilon \subseteq \tau^{MPS}(\hat{f},t_1)$ is instrumental in turning the time-balance property into a property (the lag-balance property) that allows us to compute a tighter upper bound to the total lag in Step 2.

Accordingly, also the lag-balance property, as reported in this paper in Lemma 24, happens to be an instance of a more general lag-balance property. In fact, by repeating about the same steps by which we derive the lag-balance property from the time-balance property (proof of Lemma 24), but applying these steps to the general property instead of the time-balance property, one would obtain a general lag-balance property that holds for any work-conserving scheduler.

On the opposite end, the time-balance property in Lemma 6 holds under more relaxed constraints than imposing that all the parameters and quantities in items 1-4 in Lemma 7 are fixed. In fact, for the RHS of (34) to decrease as $\sum_{v\in\Upsilon} \text{lag}_v(\hat{s})$ increases, it is enough that only all the other variables in the RHS of (34) are fixed.

The time-balance property in (34) holds under the above weaker constraints, because the bound (34) is computed assuming that the DPS provides the most unfavorable service it could provide, i.e., the service that causes the difference $\hat{s}-d_i^j$ to reach its maximum possible value. The reader will find details in the proof of Lemma 6. In particular, in that proof we exploit (implicitly) that fact that, after assuming that the above worst-case DPS service occurs, the general property holds for any work-

conserving scheduler, provided that only the parameters in item 1 in Lemma 7 and the sum $\sum_{h\in\tau(\hat{J},t_1)} \mathrm{lag}_h(t_1)$ are fixed. Unfortunately, even if the service of the DPS is fixed, the general property becomes however quite hard to visualize when only these two weaker invariants hold. We do not show the property also for this case.

7 Proof machinery

As a support to the proof of the harmonic bound, in this section we provide a proof machinery containing all the properties we need to implement the steps described in the outline in Section 5. As the definitions in Section 4, most properties basically coincide with corresponding properties of the machineries used in (Devi and Anderson (2008); Valente and Lipari (2005b)). Some properties are a little generalized with respect to (Devi and Anderson (2008); Valente and Lipari (2005b)), to prepare the ground for the use of the lag-balance property.

The machinery is a fairly rich collection of lemmas; therefore, although in the following subsections we try to describe, intuitively, the role in the proof of each property, the full picture and the contribution of every component will probably be completely clear only after putting all the pieces together in Section 8. Alternatively, the reader may skip directly to the proof in Section 8, and jump back to next lemmas as needed.

7.1 Busy interval and blocking tasks

An obvious condition for Lemma 6 to hold in Step 0 is that the time instant t_1 defined in that lemma exists. The next lemma guarantees this fact.

Lemma 8 *Given a job portion $\hat{J} \subseteq J_i^j$ blocked by priority and the release time r_i^j of J_i^j, the start time $\hat{b}$ of the busy interval for $\hat{J}$ belongs to $[r_i^j, d_i^j)$.*

Proof First, $\hat{b} \geq r_i^j$ holds by Definition 2. Second, as for the remaining inequality $\hat{b} \leq d_i^j$, we prove it by contradiction. To this purpose, suppose that the opposite, i.e., $\hat{b} > d_i^j$ holds, and let Θ denote the set of M jobs in execution at time $\hat{b}$.

By Definition 2, all the jobs in Θ have a deadline earlier than or equal to d_i^j. Hence they have all been released before time d_i^j. This fact, plus the fact that all the jobs in Θ are still pending at time $\hat{b}$ (recall that, according to our definition of *pending*, a job in execution is still pending), imply that there exists a minimal time instant $t_s \in [r_i^j, d_i^j]$, such that all the jobs in Θ are pending throughout $[t_s, \hat{b}]$. Therefore, since Θ contains M jobs, all with a deadline earlier than or equal to d_i^j, by Lemma 1 all the processors of the MPS would have been busy executing jobs with deadlines earlier than or equal to d_i^j throughout $[t_s, \hat{b}]$ (i.e., executing either jobs in Θ or jobs with an even earlier deadline). But, by Definition 2, this implies that the busy interval for $\hat{J}$ should have started at or before time t_s, which contradicts our assumption that $\hat{b} > d_i^j$ (because $t_s \leq d_i^j$). □

We conclude this subsection with the following property, which we use in various lemmas to implement Step 2.

Lemma 9 *Given a job portion $\hat{J} \subseteq J_i^j$ starting at time $\hat{s}$ after being blocked by priority, all the tasks in $\tau^{MPS}(\hat{J},t)$ are in service (i.e., have their jobs in execution) at all times t in every sub-interval of $[0,\hat{s})$ during which $|\tau^{MPS}(\hat{J},t)| \leq M$ holds, whereas only tasks in $\tau^{MPS}(\hat{J},t)$ are in service at all times t in every sub-interval of $[0,\hat{s})$ during which $|\tau^{MPS}(\hat{J},t)| > M$ holds.*

Proof The thesis trivially follows from Lemma 1 and the fact that the tasks in $\tau^{MPS}(\hat{J},t)$ own, of course, the portions with the earliest deadlines among all the portions pending during any sub-interval of $[0,\hat{s})$. □

7.2 Dedicated-Processor System (DPS)

The next lemma states the main property of the DPS that allows us to prove Lemma 4 in Step 1. Informally, this property says that the maximum tardiness of a job grows with the difference between the total work that the MPS may do while the last portion of the job waits to be started, and the total work that the DPS has to do while executing the same job. The link between this property and Lemma 4 is that, as we also prove, this difference is in its turn upper-bounded by a quantity that grows with $\Lambda(k)$.

Before reporting the lemma, we stress two points. First, instead of providing directly an upper bound to $f_i^j - d_i^j$, the lemma provides an upper bound to $\hat{s} - d_i^j$ for a generic job portion $\hat{J}$, where $\hat{s}$ is the start time of $\hat{J}$. This intermediate result will come in handy for proving the time-balance property in Step 0. Second, the lemma refers to two time intervals that start from a generic time instant $t_1 \in [\hat{b}, \min\{d_i^j, \hat{s}\})$ (where the upper limit $\min\{d_i^j, \hat{s}\}$ for t_1 is needed to let both time intervals be well-defined, as explained in the proof). As we already discussed in the description of Step 0, we exploit this degree of freedom to get a tighter upper bound to the total lag in Step 2.

Lemma 10 *Given a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, the busy interval $[\hat{b},\hat{s})$ for $\hat{J}$, and any time instant $t_1 \in [\hat{b}, \min\{d_i^j, \hat{s}\})$, we have*

$$\hat{s} - d_i^j \leq \frac{\sum_{h \in \tau} W_h^{MPS}(t_1,\hat{s}) - \sum_{h \in \tau} W_h^{DPS}(t_1, d_i^j)}{M}, \tag{35}$$

where: $W_h^{MPS}(t_1,\hat{s}) \stackrel{(1)}{=} W_h^{MPS}(\hat{s}) - W_h^{MPS}(t_1)$, $W_h^{DPS}(t_1,d_i^j) \stackrel{(1)}{=} W_h^{DPS}(d_i^j) - W_h^{DPS}(t_1)$, and the quantity M is measured in service units per time unit (i.e., is a speed).

Proof First, by Lemma 8, t_1 exists. Second, both time intervals $[t_1,\hat{s}]$ and $[t_1,d_i^j]$ are well-defined, because $t_1 \leq \hat{s}$ and $t_1 \leq d_i^j$ hold by definition of t_1. Then, considering the following trivial equality: $\hat{s} - d_i^j = (\hat{s} - t_1) - (d_i^j - t_1)$, we prove the thesis by: 1) computing $\hat{s} - t_1$ as a function of the total work done by the MPS during $[t_1,\hat{s})$, i.e., of the sum $\sum_{h \in \tau} \left[W_h^{MPS}(\hat{s}) - W_h^{MPS}(t_1)\right]$, 2) computing a lower bound to $d_i^j - t_1$ as a function of the total work done by the DPS during $[t_1,d_i^j]$, i.e., of the sum

$\sum_{h\in\tau}\left[W_h^{DPS}(d_i^j)-W_h^{DPS}(t_1)\right]$, and 3) subtracting this lower bound from the value computed for $\hat{s}-t_1$ in the first step.

As for the first step, all processors of the MPS are constantly busy throughout $[t_1,\hat{s})$, by definition of busy interval, and because $[t_1,\hat{s})\subseteq[b,\hat{s})$. Therefore, the MPS works at constant total speed M during $[t_1,\hat{s})$. It follows that

$$\hat{s}-t_1=\frac{\sum_{h\in\tau}\left[W_h^{MPS}(\hat{s})-W_h^{MPS}(t_1)\right]}{M}\stackrel{(1)}{=}\frac{\sum_{h\in\tau}W_h^{MPS}(t_1,\hat{s})}{M}, \tag{36}$$

where the denominator is measured of course in service units per time unit.

Regarding the second step, and considering that, by Definition 4, the maximum total speed of the DPS is $\sum_{h\in\tau}U_h$, with $\sum_{h\in\tau}U_h\leq M$, we get

$$d_i^j-t_1\geq\frac{\sum_{h\in\tau}\left[W_h^{DPS}(d_i^j)-W_h^{DPS}(t_1)\right]}{M}\stackrel{(1)}{=}\frac{\sum_{h\in\tau}W_h^{DPS}(t_1,d_i^j)}{M}. \tag{37}$$

Subtracting (37) from (36), we get the thesis. □

Note that we could correctly replace M with $\sum_{h\in\tau}U_h$ in (37), and obtain a tighter upper bound than (35) for the case $\sum_{h\in\tau}U_h<M$. Instead, we use M in (37) to get simpler formulas.

7.3 Lag

The next lemma provides us with two sufficient conditions for the quantity $\mathrm{lag}_h(t)$ to be equal to just the difference between the amounts of service received by the task τ_h in the DPS and in the MPS. We use this property in almost all lemmas related to lags.

Lemma 11 *Given a job portion $\hat{J}\subseteq J_i^j$, if a task τ_h belongs to $\tau^{MPS}(\hat{J},t)$ at time t, or, more in general, if only jobs of τ_h with a deadline earlier than or equal to d_i^j have been executed by time t, then*

$$W_h^{MPS-B}\langle d_i^j\rangle(t)=W_h^{MPS}(t), \tag{38}$$

which implies $lag_h(t)=W_h^{DPS}(t)-W_h^{MPS}(t)$.

Proof We start by proving the thesis for the more general case where the MPS has not yet executed any job of τ_h with a deadline strictly later than d_i^j by time t. This implies $W_h^{MPS}(t)\leq\sum_{n=1}^{m}l_h^n$ with m defined as in Definition 5. Then (38) holds from (21). Therefore,

$$\mathrm{lag}_h(t)\stackrel{(23)}{=}\mathrm{lag}_h\langle d_i^j\rangle(t)\stackrel{(22)}{=}W_h^{DPS}(t)-W_h^{MPS-B}(t)\stackrel{(38)}{=}W_h^{DPS}(t)-W_h^{MPS}(t). \tag{39}$$

As for the special case where τ_h belongs to $\tau^{MPS}(\hat{J},t)$ at time t, by Definition 3, the task has at least one pending job with a deadline earlier than or equal to d_i^j at time t. This implies that by time t the MPS has not yet executed any job of τ_h with a deadline strictly later than d_i^j. Then (38) and hence (39) hold also in this case. □

The next lemma tells us that, given a portion $\hat{J} \subseteq J_i^j$, the lag $\text{lag}_h(t)$ of a task belonging to $\tau^{MPS}(\hat{J},t)$ does not increase while the task is served in the MPS. We use this property in two other lemmas in Step 2.

Lemma 12 *Given a job portion $\hat{J} \subseteq J_i^j$, the lag $lag_h(t)$ of a task τ_h decreases by at least $(1-U_i)(t_2-t_1)$ service units in a time interval $[t_1,t_2]$ during which the task continuously belongs to $\tau^{MPS}(\hat{J},t)$ and is continuously served in the MPS.*

Proof During $[t_1,t_2]$, the task receives an amount of service $W_h^{MPS}(t_2)-W_h^{MPS}(t_1) = t_2-t_1$ in the MPS. On the other side, the amount of service $W_h^{DPS}(t_2)-W_h^{DPS}(t_1)$ given to the task in the DPS is at most $U_i \cdot (t_2-t_1)$. In addition, since τ_h continuously belongs to $\tau^{MPS}(\hat{J},t)$ during $[t_1,t_2]$, $\text{lag}_h(t) = W_h^{DPS}(t) - W_h^{MPS}(t)$ holds at all t in $[t_1,t_2]$ by Lemma 11. Using these relations, we can write

$$\begin{aligned} \text{lag}_h(t_2) - \text{lag}_h(t_1) = &\left[W_h^{DPS}(t_2) - W_h^{MPS}(t_2)\right] - \left[W_h^{DPS}(t_1) - W_h^{MPS}(t_1)\right] = \\ &\left[W_h^{DPS}(t_2) - W_h^{DPS}(t_1)\right] - \left[W_h^{MPS}(t_2) - W_h^{MPS}(t_1)\right] \leq \\ &U_i \cdot (t_2-t_1) - (t_2-t_1) = (U_i-1)(t_2-t_1) = -(1-U_i)(t_2-t_1). \end{aligned} \tag{40}$$

□

The following two lemmas link the maximum lag of a task, on the start time of a generic job portion, with how late that portion is. In the first lemma, we assume that the length of the job the portion belongs to is maximum. As we explain in more detail in Section 7.7, this assumption helps us simplify proofs in Step 2.

Lemma 13 *Given a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, and a job portion $\hat{J}_h \subseteq J_h^l$ starting at time s_h, and denoting by $\hat{l}_h$ the size, in service units, of the part of the job J_h^l not yet executed in the MPS by time s_h, we have that, if: 1) the job J_h^l has the maximum possible length, i.e., $l_h^l = L_h$, 2) τ_h belongs to $\tau^{MPS}(\hat{J},s_h)$, and 3) $s_h \geq r_h^l$, then*

$$lag_h(s_h) \leq (s_h - d_h^l)U_h + \hat{l}_h. \tag{41}$$

Proof To prove the thesis, we prove, firstly, two preliminary relations. For both relations, we use that fact that, since $l_h^l = L_h$, the DPS continuously executes J_h^l during $[r_h^l, d_h^l]$, by Definition 4. This implies that, by time d_h^l, the DPS has finished executing J_h^l and no subsequent job of τ_h. Instead, being $\hat{l}_h$ the part of J_h^l not yet executed by the MPS by time s_h, it follows

$$\hat{l}_h = W_h^{DPS}(d_h^l) - W_h^{MPS}(s_h). \tag{42}$$

As for the second relation, we consider two alternatives. First, $s_h \leq d_h^l$. In this case, from $s_h \geq r_h^l$ it follows that $[s_h, d_h^l] \subseteq [r_h^l, d_h^l]$ and hence the DPS continuously executes J_h^l during $[s_h, d_h^l]$. This implies $W_h^{DPS}(d_h^l) - W_h^{DPS}(s_h) = (d_h^l - s_h)U_h$, which, reversed, yields $W_h^{DPS}(s_h) - W_h^{DPS}(d_h^l) = (s_h - d_h^l)U_h$. If, instead, $s_h > d_h^l$, then the DPS may not serve the task τ_h continuously during $[d_h^l, s_h]$. It follows that

$$W_h^{DPS}(s_h) - W_h^{DPS}(d_h^l) \leq (s_h - d_h^l)U_h. \tag{43}$$

In the end, the last inequality holds for any ordering between s_h and d_h^l. We can now compute our bound to $\text{lag}_h(\hat{s})$ with the following derivations, where the first equality follows from Lemma 11 and the fact that τ_h belongs to $\tau^{MPS}(\hat{J}, s_h)$,

$$\begin{aligned} \text{lag}_h(s_h) = W_h^{DPS}(s_h) - W_h^{MPS}(s_h) &= \\ W_h^{DPS}(d_h^l) - W_h^{DPS}(d_h^l) + W_h^{DPS}(s_h) - W_h^{MPS}(s_h) &\overset{\text{rearranging}}{=} \\ \left[W_h^{DPS}(s_h) - W_h^{DPS}(d_h^l)\right] + \left[W_h^{DPS}(d_h^l) - W_h^{MPS}(s_h)\right] &\overset{(43)}{\leq} \\ (s_h - d_h^l)U_h + \left[W_h^{DPS}(d_h^l) - W_h^{MPS}(s_h)\right] &\overset{(42)}{=} \\ (s_h - d_h^l)U_h + \hat{I}. & \end{aligned} \tag{44}$$

□

Lemma 14 *Given a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, and a job portion $\hat{J}_h \subseteq J_h^l$ that starts at time s_h, with $s_h = r_h^l$ (which implies that $\hat{J}_h$ is the first portion of J_h^l), we have that, if τ_h belongs to $\tau^{MPS}(\hat{J}, s_h)$, then $lag_h(s_h) = 0$ holds.*

Proof First, if $l = 1$ also holds, i.e., J_h^l is the very first job of τ_h, then either the MPS or DPS have not yet executed any job of τ_h by time s_h. In the end, $W_h^{DPS}(s_h) = W_h^{MPS}(s_h) = 0$. Otherwise, if $l > 1$, then the DPS has finished J_h^{l-1} at time s_h, and is just starting to execute J_h^l. But, this is exactly the case also for the MPS. Therefore, by time s_h the MPS has provided τ_h with exactly the same amount of service as the DPS, i.e., $W_h^{DPS}(s_h) = W_h^{MPS}(s_h)$ holds. The thesis then follows from Lemma 11, because τ_h belongs to $\tau^{MPS}(\hat{J}, s_h)$. □

The last property of the lag of a task that we use in Step 2 is concerned with the possible values of this function at the time instants at which the task enters or exits the sets $\tau^{MPS}(\hat{J}, t)$ and $\tau(\hat{J}, t)$.

Lemma 15 *Given a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, we have that, if t_e is a time instant at which a task τ_e enters the set $\tau^{MPS}(\hat{J}, t)$ or the set $\tau(\hat{J}, t)$, then $lag_e(t_e) = 0$. If, instead, the task τ_e exits $\tau(\hat{J}, t)$ at time t_e, then $lag_e(t_e) > 0$.*

Proof As for the case of a task entering $\tau^{MPS}(\hat{J}, t)$ or $\tau(\hat{J}, t)$, according to Definitions 6 and 7, a task can enter $\tau(\hat{J}, t)$ only by entering $\tau^{MPS}(\hat{J}, t)$. Then, focusing on a task that enters $\tau^{MPS}(\hat{J}, t)$ at time t_e, we can note, first, that it cannot happen because of a job completion in the MPS. In fact, when a job is completed in the MPS, either the task has no more pending jobs in the MPS, or the next job to execute for the task has a strictly later deadline than the just completed one. Second, if a task already has pending jobs right before time t_e in the MPS, then the task cannot enter $\tau^{MPS}(\hat{J}, t)$ because of the arrival of a new job at time t_e either. In fact, the newly-released job has a strictly later deadline than the pending ones.

In the end, the only possibility for a task τ_e to enter $\tau^{MPS}(\hat{J}, t)$ at time t_e is that the task has no pending job in the MPS right before time t_e, and a new job of the task is released at time t_e. Since the DPS has certainly finished all previous jobs of the task by time t_e too (by Definition 4), it follows that $W_e^{DPS}(t_e) = W_e^{MPS}(t_e)$. This implies $\text{lag}_e(t_e) = 0$ by Lemma 11.

As for the exit from $\tau(\hat{J},t)$, we can consider two alternatives. First, it happens because the task τ_e exits $\tau^{MPS}(\hat{J},t)$ at time t_e. In this case, by Definitions 6 and 7, the only possibility for the task to not enter $\tau^{DPS}(\hat{J},t_e^+)$, and hence to exit also $\tau(\hat{J},t)$, is that $\text{lag}_e(t_e) > 0$. The second alternative is that τ_e exits $\tau(\hat{J},t)$ for a different reason than because the task τ_e exits $\tau^{MPS}(\hat{J},t)$ at time t_e. In this second case, the only possibility for the task to exit $\tau(\hat{J},t)$ at time t_e is that the task belongs to $\tau^{DPS}(\hat{J},t_e)$ and exits the latter set at time t_e. For this to happen, $\text{lag}_e(t_e) > 0$ must hold, again by Definition 6. □

7.4 Total lag

In Section 4.5 we highlighted that, according to Figures 1.D and 1.F, the number of tasks in $\tau^{MPS}(J_2^2,t)$ is lower than the total utilization of the task set while the total lag grows. This is a concrete example of the general property stated in the following lemma.

Lemma 16 *If the total lag for a job portion $\hat{J} \subseteq J_i^j$ blocked by priority is strictly increasing at all t in $[t_a,t_b)$, then $|\tau^{MPS}(\hat{J},t)| \leq \lceil U_{sum} \rceil - 1$ holds for all t in $[t_a,t_b)$.*

Proof To prove the thesis, we start by considering that the total lag cannot increase when the set $\tau(\hat{J},t)$ changes. In fact, by Lemma 15, if a task enters $\tau(\hat{J},t)$, then its lag is zero, while if a task exits $\tau(\hat{J},t)$, then its lag is positive.

Consider then any sub-interval $[t_1,t_2) \subseteq [t_a,t_b)$ such that neither the set $\tau(\hat{J},t)$ or the set $\tau^{MPS}(\hat{J},t)$ change at any time in $[t_1,t_2)$. If $\tau(\hat{J},t)$ changes at time t_2, then, as above noted, the total lag cannot increase at time t_2. In addition, if only $\tau^{MPS}(\hat{J},t)$ changes at time t_2, then the total lag does not change at all, because the total lag is a continuous function in all time intervals during which $\tau(\hat{J},t)$ does not change (as lags are continuous functions). It follows that, if the thesis holds for any sub-interval $[t_1,t_2)$ defined as above, then the thesis holds also for the whole time interval $[t_a,t_b)$. Since $\tau(\hat{J},t) = \tau(\hat{J},t_1)$ holds for all $t \in [t_1,t_2)$, and defining, for brevity, $W_h^{MPS-B}(t) \equiv W_h^{MPS-B}\langle d_i^j \rangle(t)$, the variation of the total lag during $[t_1,t_2)$ is equal to

$$
\begin{aligned}
& \textstyle\sum_{h\in\tau(\hat{J},t_1)} [\text{lag}_h(t_2) - \text{lag}_h(t_1)] && \overset{(22)}{=} \\
& \textstyle\sum_{h\in\tau(\hat{J},t_1)} \left[(W_h^{DPS}(t_2) - W_h^{MPS-B}(t_2)) - (W_h^{DPS}(t_1) - W_h^{MPS-B}(t_1))\right] && = \\
& \textstyle\sum_{h\in\tau(\hat{J},t_1)} \left[(W_h^{DPS}(t_2) - W_h^{DPS}(t_1)) - (W_h^{MPS-B}(t_2) - W_h^{MPS-B}(t_1))\right] && = \\
& \textstyle\sum_{h\in\tau(\hat{J},t_1)} (W_h^{DPS}(t_2) - W_h^{DPS}(t_1)) + \\
& \textstyle -\sum_{h\in\tau(\hat{J},t_1)} (W_h^{MPS-B}(t_2) - W_h^{MPS-B}(t_1)) && \overset{\text{Def. 4}}{\leq} \\
& \textstyle\sum_{h\in\tau(\hat{J},t_1)} U_h \cdot (t_2 - t_1) - \sum_{h\in\tau(\hat{J},t_1)} (W_h^{MPS-B}(t_2) - W_h^{MPS-B}(t_1)) && \overset{\tau(\hat{J},t_1)\subseteq\tau}{\leq} \\
& U_{sum} \cdot (t_2 - t_1) - \textstyle\sum_{h\in\tau(\hat{J},t_1)} (W_h^{MPS-B}(t_2) - W_h^{MPS-B}(t_1)) && \leq \\
& U_{sum} \cdot (t_2 - t_1) - \textstyle\sum_{h\in\tau^{MPS}(\hat{J},t_1)} (W_h^{MPS-B}(t_2) - W_h^{MPS-B}(t_1)), &&
\end{aligned}
\tag{45}
$$

where the last inequality follows from $\tau^{MPS}(\hat{J},t_1) \subseteq \tau(\hat{J},t_1)$ and $W_h^{MPS-B}(t_2) - W_h^{MPS-B}(t_1) \geq 0$ for all $h \in \tau(\hat{J},t_1)$ (by (21) and because $W_h^{MPS}(t_2) - W_h^{MPS}(t_1) \geq 0$).

As for the sum in the last line in (45), by Lemma 11, we have:

$$\forall \tau_h \in \tau^{MPS}(\hat{J},t_1) \quad W_h^{MPS-B}(t) = W_h^{MPS}(t). \tag{46}$$

In addition, by Lemma 9, a number of processors at least equal to $|\tau^{MPS}(\hat{J},t_1)|$ is busy serving tasks in $\tau^{MPS}(\hat{J},t_1)$ during $[t_1,t_2)$. Since every processor of the MPS has unit speed, it follows that

$$\sum_{h \in \tau^{MPS}(\hat{J},t_1)} (W_h^{MPS}(t_2) - W_h^{MPS}(t_1)) \geq \min\left[|\tau^{MPS}(\hat{J},t_1)|, M\right] \cdot (t_2 - t_1). \tag{47}$$

Replacing (46) and (47) in (45), we get

$$\sum_{h \in \tau(\hat{J},t_1)} [\mathrm{lag}_h(t_2) - \mathrm{lag}_h(t_1)] \leq \left\{U_{sum} - \min\left[|\tau^{MPS}(\hat{J},t_1)|, M\right]\right\} \cdot (t_2 - t_1). \tag{48}$$

Since $U_{sum} \leq M$, the only possibility for the RHS of (48) to be strictly positive is that $|\tau^{MPS}(\hat{J},t_1)| < U_{sum}$. In this respect, being $|\tau^{MPS}(\hat{J},t_1)|$ an integral number, the last relation implies that $|\tau^{MPS}(\hat{J},t_1)|$ can be equal at most to the largest integer strictly lower than U_{sum}, i.e., that $|\tau^{MPS}(\hat{J},t_1)| \leq \lceil U_{sum} \rceil - 1$. □

7.5 An upper bound to the extra work done by the MPS with respect to the DPS

The following lemma states that the total lag is an upper bound to the difference between the total work done by the MPS and the total work done by the DPS during the two time intervals considered in Lemma 10. This result, combined with Lemma 10 itself, allows us to prove the bound (31) in Step 1.

Actually, the lemma provides a relation between two more general quantities than just two total amounts of work. In fact, the relation concerns the total service given to any subset of tasks in τ. This generalization, plus the fact that, as in Lemma 10, the two time intervals of interest start from a generic time instant $t_1 \in [\hat{b}, \min\{d_i^j, \hat{s}\})$, are instrumental in exploiting the lag-balance property to get a tighter upper bound to the total lag in Step 2. The proof of the lemma is relatively long, so, for ease of exposition, we state the lemma first, and then: (a) introduce the idea behind the proof, (b) present the per-task bounds used to prove the lemma as a separate lemma, (c) provide the proof.

Lemma 17 *Given: (a) a portion $\hat{J} \subseteq J_i^j$ blocked by priority, (b) the busy interval $[\hat{b},\hat{s})$ for $\hat{J}$, (c) any time instant $t_1 \in [\hat{b}, \min\{d_i^j, \hat{s}\})$, and (d) any set of tasks Υ that does not contain τ_i, we have*

$$\sum_{h \in \tau \setminus \Upsilon} W_h^{MPS}(t_1,\hat{s}) - \sum_{h \in \tau \setminus \Upsilon} W_h^{DPS}(t_1,d_i^j) \leq \sum_{h \in \tau(\hat{J},t_1) \setminus \Upsilon} lag_h(t_1) - \hat{l}, \tag{49}$$

where: $W_h^{MPS}(t_1,\hat{s}) \stackrel{(1)}{=} W_h^{MPS}(\hat{s}) - W_h^{MPS}(t_1)$, $W_h^{DPS}(t_1,d_i^j) \stackrel{(1)}{=} W_h^{DPS}(d_i^j) - W_h^{DPS}(t_1)$ and $\hat{l}$ is the size, in service units, of the part of the job J_i^j not yet executed in the MPS by time $\hat{s}$.

To compute an upper bound to the left-hand side (LHS) of (49), we focus on a set of tasks with the following two properties: (1) it contains at least all the tasks, except for those in Υ, that are served by the MPS during $[t_1,\hat{s}]$, and (2) for every task in the set, it is possible to lower-bound the service that the task receives in the DPS during $[t_1,d_i^j]$ as a function of the service that it receives in the MPS during $[t_1,\hat{s}]$. As for the first property, to compute the value of the first sum in the LHS of (49) it is necessary to consider at least all the tasks served during $[t_1,\hat{s}]$, except for those in Υ. Instead, the second property allows us to lower-bound the value of the second sum in the LHS of (49) as a function of the first sum. In particular, it allows us to prove that the difference between the two sums is upper-bounded exactly by the RHS of (49). To define such a set, we start from the following superset.

Definition 12 Referring to the notations in Lemma 17, we define as $\overline{\tau}$ the union of the set of tasks served during $[t_1,\hat{s}]$ in the MPS and of the set $\tau(\hat{J},t_1)$.

The set we need is $\overline{\tau}\setminus\Upsilon$, where Υ is defined as in Lemma 17. In fact, as for the first property, the following equality trivially holds:

$$\sum_{h\in\tau\setminus\Upsilon} W_h^{MPS}(t_1,\hat{s}) = \sum_{h\in\overline{\tau}\setminus\Upsilon} W_h^{MPS}(t_1,\hat{s}). \tag{50}$$

On the other hand, the second property follows from the fact that the superset $\overline{\tau}$ itself enjoys the second property, as stated in the following lemma.

Lemma 18 *Given: (a) a portion $\hat{J}\subseteq J_i^j$ blocked by priority, (b) the busy interval $[\hat{b},\hat{s})$ for $\hat{J}$, (c) any time instant $t_1\in[\hat{b},\min\{d_i^j,\hat{s}\})$, and (d) the set $\overline{\tau}$ in Definition 12, we have*

$$\forall\tau_h\in\overline{\tau}\quad W_h^{MPS}(t_1,\hat{s})\le W_h^{DPS}(t_1,d_i^j)+lag_h(t_1). \tag{51}$$

In addition, $\tau_i\in\overline{\tau}$, and

$$W_i^{MPS}(t_1,\hat{s})\le W_i^{DPS}(t_1,d_i^j)+lag_i(t_1)-\hat{l}, \tag{52}$$

where $\hat{l}$ is the size, in service units, of the part of the job J_i^j not yet executed in the MPS by time $\hat{s}$.

Proof First, as we demonstrated at the beginning of the proof of Lemma 10, both time intervals $[t_1,\hat{s}]$ and $[t_1,d_i^j]$ are well-defined. In addition, J_i^j is pending at time t_1, by definition of t_1 and by Lemma 8. Therefore τ_i belongs to $\tau^{MPS}(\hat{J},t_1)$, and hence to $\tau(\hat{J},t_1)$. This implies that τ_i belongs to $\overline{\tau}$, because $\tau(\hat{J},t_1)\subseteq\overline{\tau}$. Using this fact, we prove the rest of the thesis by considering, separately, (a) the tasks in $\overline{\tau}\setminus\{\tau_i\}$ that receive some service in the MPS during $[t_1,\hat{s})$, (b) the task τ_i itself, and (c) the tasks in $\overline{\tau}\setminus\{\tau_i\}$ that do not receive any service in the MPS during $[t_1,\hat{s})$.

As for the first subset, consider any task $\tau_h\in\overline{\tau}\setminus\{\tau_i\}$, for which some jobs are executed, completely or in part, by the MPS during $[t_1,\hat{s})$. These jobs have deadlines earlier than or equal to d_i^j. This implies that, by time t_1, the MPS has not yet executed any job of τ_h with a deadline later than d_i^j. Therefore, by Lemma 11 we have

$$W_h^{MPS-B\langle d_i^j\rangle}(t_1)=W_h^{MPS}(t). \tag{53}$$

Moreover, the fact that the jobs of τ_h executed during $[t_1, \hat{s})$ have deadlines earlier than or equal to d_i^j also implies that the DPS must have completed all of these jobs by time d_i^j. Of course, by time d_i^j, the DPS may have executed, completely or in part, also other jobs of τ_h. In the end, the following inequality holds:

$$W_h^{MPS}(\hat{s}) \leq W_h^{DPS}(d_i^j). \tag{54}$$

Using the last two relations, we can write:

$$\begin{aligned} W_h^{MPS}(t_1,\hat{s}) &\overset{(1)}{=} W_h^{MPS}(\hat{s}) - W_h^{MPS}(t_1) \overset{(54)}{\leq} W_h^{DPS}(d_i^j) - W_h^{MPS}(t_1) \;= \\ & W_h^{DPS}(d_i^j) - W_h^{MPS}(t_1) + W_h^{DPS}(t_1) - W_h^{DPS}(t_1) \;= \\ & \left[W_h^{DPS}(d_i^j) - W_h^{DPS}(t_1)\right] + \left[W_h^{DPS}(t_1) - W_h^{MPS}(t_1)\right] \overset{(1)}{=} \\ & W_h^{DPS}(t_1,d_i^j) + \left[W_h^{DPS}(t_1) - W_h^{MPS}(t_1)\right] \overset{(53)}{=} \\ & W_h^{DPS}(t_1,d_i^j) + (W_h^{DPS}(t_1) - W_h^{MPS-B}\langle d_i^j\rangle(t_1)) \overset{(22)}{=} \\ & W_h^{DPS}(t_1,d_i^j) + \mathrm{lag}_h\langle d_i^j\rangle(t_1) \overset{(23)}{=} W_h^{DPS}(t_1,d_i^j) + \mathrm{lag}_h(t_1). \end{aligned} \tag{55}$$

Consider now τ_i. The DPS must of course have finished J_i^j by time d_i^j, whereas the MPS has still to execute at least $\hat{l}$ service units of that job at time $\hat{s}$, i.e., $W_i^{MPS}(\hat{s}) \leq W_i^{DPS}(d_i^j) - \hat{l}$ holds. Setting $h = i$ in (55), and replacing the last inequality instead of (54), we get (52).

We are left now with the tasks in $\overline{\tau} \setminus \{\tau_i\}$ whose jobs are not executed at all by the MPS during $[t_1, \hat{s})$. If τ_h is one such task, we have $W_h^{MPS}(t_1,\hat{s}) = 0$. In addition, if $\mathrm{lag}_h(t_1) \geq 0$ holds for the task, then $W_h^{DPS}(t_1,d_i^j) + \mathrm{lag}_h(t_1) \geq 0$ trivially holds. The last inequality implies that (51) holds as well, because $W_h(t_1,\hat{s}) = 0$.

On the other hand, in the case where $\mathrm{lag}_h(t_1) < 0$ holds, we can consider that, by (23), (22) and (21), $\mathrm{lag}_h(t_1) \geq W_h^{DPS}(t_1) - \sum_{n=1}^{m} l_h^n$, i.e., $|\mathrm{lag}_h(t_1)|$ is equal at most to the sum of the sizes of the job portions, with a deadline earlier than or equal to d_i^j, that the DPS has still to execute at time t_1. This sum is in its turn at most equal to $W_h^{DPS}(t_1,d_i^j)$, because during $[t_1,d_i^j]$ the DPS must finish both these job portions and possible other jobs of τ_h, released during $[t_1,d_i^j)$ and with a deadline earlier than or equal to d_i^j. In the end, also in this case, $W_h^{DPS}(t_1,d_i^j) + \mathrm{lag}_h(t_1) \geq 0$ holds, which, combined with $W_h(t_1,\hat{s}) = 0$, proves (51) again. □

Of course $\tau(\hat{J},t_1)$ and hence $\overline{\tau}$ may not contain all the tasks that receive service in the DPS during $[t_1,d_i^j]$. Thus the bounds (51) and (52) may be loose in some scenarios. This is not however a problem, because we are interested in worst-case scenarios. Using the relations proved so far, we can now prove Lemma 17.

Proof (Lemma 17) Considering that, by Lemma 18, $\overline{\tau}$ contains τ_i, we have

$$\begin{aligned}
\sum_{h\in\tau\setminus\Upsilon} W_h^{MPS}(t_1,\hat{s}) &\overset{(50)}{=} \sum_{h\in\overline{\tau}\setminus\Upsilon} W_h^{MPS}(t_1,\hat{s}) \overset{\text{rearranging}}{=} \\
&\sum_{h\in\overline{\tau}\setminus(\Upsilon\cup\{i\})} W_h^{MPS}(t_1,\hat{s}) + W_i^{MPS}(t_1,\hat{s}) \overset{(51)+(52)}{\leq} \\
\sum_{h\in\overline{\tau}\setminus(\Upsilon\cup\{i\})} \left(W_h^{DPS}(t_1,d_i^j)+\text{lag}_h(t_1)\right) + W_i^{DPS}(t_1,d_i^j) + \text{lag}_i(t_1) - \hat{l} &\overset{\text{rearranging}}{=} \\
\sum_{h\in\overline{\tau}\setminus\Upsilon} W_h^{DPS}(t_1,d_i^j) + \sum_{h\in\overline{\tau}\setminus\Upsilon} \text{lag}_h(t_1) - \hat{l} &\overset{\overline{\tau}\subseteq\tau}{\leq} \\
\sum_{h\in\tau\setminus\Upsilon} W_h^{DPS}(t_1,d_i^j) + \sum_{h\in\overline{\tau}\setminus\Upsilon} \text{lag}_h(t_1) - \hat{l} &\overset{\tau(\hat{J},t_1)\subseteq\overline{\tau}}{=} \\
\sum_{h\in\tau\setminus\Upsilon} W_h^{DPS}(t_1,d_i^j) + \sum_{h\in\tau(\hat{J},t_1)\setminus\Upsilon} \text{lag}_h(t_1) + & \\
+ \sum_{h\in(\overline{\tau}\setminus\tau(\hat{J},t_1))\setminus\Upsilon} \text{lag}_h(t_1) - \hat{l}. &
\end{aligned} \tag{56}$$

From (56), we have that the thesis holds if $\sum_{h\in(\overline{\tau}\setminus\tau(\hat{J},t_1))\setminus\Upsilon} \text{lag}_h(t_1) \leq 0$ holds. We prove the latter inequality by proving that $\text{lag}_h(t_1) \leq 0$ holds for every task τ_h in $(\overline{\tau}\setminus\tau(\hat{J},t_1))\setminus\Upsilon$. To this purpose, we prove, as a first step and by contradiction, that τ_h cannot have pending jobs at time t_1. Suppose then that τ_h has some pending job in the MPS at time t_1. It follows that one or more of these pending jobs are executed, at least in part, during $[t_1,\hat{s})$. In fact, by definition of $\overline{\tau}$, for τ_h to belong to $(\overline{\tau}\setminus\tau(\hat{J},t_1))\setminus\Upsilon$, the task must receive some service during $[t_1,\hat{s})$. But the deadline of the jobs executed during $[t_1,\hat{s})$ has to be earlier than or equal to d_i^j, by Definition 2. This implies that τ_h belongs to $\tau^{MPS}(\hat{J},t_1)$ and therefore to $\tau(\hat{J},t_1)$ as well, by Definitions 3 and 7. This contradicts the fact that τ_h belongs to $(\overline{\tau}\setminus\tau(\hat{J},t_1))\setminus\Upsilon$.

Now, since τ_h has no pending job in the MPS at time t_1, then $W_h^{MPS}(t_1) \geq W_h^{DPS}(t_1)$ holds. In addition, we can consider that, according to the above arguments, jobs of τ_h with a deadline earlier than or equal to d_i^j are executed during $[t_1,\hat{s})$. Since these jobs are released after time t_1, it follows that no job with a strictly later deadline than d_i^j may have been released by time t_1. Therefore, at time t_1 the DPS can have served only jobs with a deadline earlier than or equal to d_i^j. From this fact and the inequality $W_h^{MPS}(t_1) \geq W_h^{DPS}(t_1)$ it follows that $\text{lag}_h(t_1) \leq 0$ by (22). □

7.6 Busy intervals are subsets of non-growing-Lag intervals

By comparing figures 1.C and 1.G, we can see that the busy interval for each portion blocked by priority happens to be a sub-interval of the last non-growing-Lag interval for the same portion. This is not an accident, but a general property, as stated in the following lemma.

Lemma 19 *The busy interval $[\hat{b},\hat{s})$ for a job portion $\hat{J}$ blocked by priority is a sub-interval of the last non-growing-Lag interval for that portion. In particular, if the last non-growing-Lag interval for $\hat{J}$ is $[b_k,f_k)$, then $\hat{b} \geq b_k$, $f_k = \hat{s}$ and*

$$\forall t_1 \in [\hat{b},\hat{s}) \qquad \sum_{h\in\tau(\hat{J},t_1)} lag_h(t_1) \leq \sum_{h\in\tau(\hat{J},b_k)} lag_h(b_k). \tag{57}$$

Proof Since the last non-growing Lag interval for $\hat{J}$ must finish before time $\hat{s}$ by Definition 9, we have that, if (57) holds, then the first part of the lemma holds, i.e., $[\hat{b},\hat{s}) \subseteq [b_k, f_k)$ and $f_k = \hat{s}$. To prove (57), we can consider that, by definition of busy interval, $|\tau^{MPS}(\hat{J},t_1)| \geq M > \lceil U_{sum} \rceil - 1$ holds for all t_1 in $[\hat{b},\hat{s})$. Therefore, by Lemma 16, (57) holds for all t_1 in $[\hat{b},\hat{s})$. □

Thanks to the last lemma, the total lag at any time instant in a busy interval is not higher than the total lag at the beginning of the non-growing-Lag interval the busy interval belongs to. This is a necessary condition in our proof of an upper bound to $\Lambda(k)$ in Step 2.

7.7 Reduced set of tasks used to upper-bound the total lag in Step 2

In this subsection we report the properties of the set $\hat{\tau}$ defined in Definition 11, plus several abbreviations and assumptions that help simplify the proof of an upper bound to $\Lambda(k)$ in Step 2. We start by the following lemma.

Lemma 20 *Suppose that the k-th non-growing-Lag interval is a non-growing-Lag interval for a job portion $\hat{J} \subseteq J_i^j$ blocked by priority.*

1. *If $b_k = 0$[8] or $k = 1$, then $\hat{\tau} = \emptyset$ and hence $\sum_{h \in \tau(\hat{J},b_k)} lag_h(b_k) \leq 0$.*
2. *Otherwise, defining as $\tilde{b}_q$ the beginning of the growing-Lag interval for $\hat{J}$ that terminates at time b_k,*
   (a) *for every task τ_g in $\hat{\tau}$, there exists a minimal time instant s_g such that $s_g \leq \tilde{b}_q$, and τ_g is served in the MPS at all times t in $[s_g, b_k)$ and belongs to $\tau^{MPS}(\hat{J},t)$ at all times t in $[s_g, b_k]$;*
   (b) *denoting by $\hat{J}_g \subseteq J_g^l$ the head of the chain of τ_g that starts at time s_g, we have that $s_g > r_g^l$ holds (r_g^l is the release time of J_g^l), and $\hat{J}_g$ is blocked by priority;*
   (c) *the set $\hat{\tau}$ is made of at most $\lceil U_{sum} \rceil - 1$ tasks.*

Proof As for item 1, if $b_k = 0$ the thesis holds trivially. Instead, we prove the thesis for $k = 1$ after proving item 2.a. Regarding item 2.a, consider any task $\tau_g \in \hat{\tau}$. By Definitions 6 and 7, for τ_g to belong to $\hat{\tau} \subseteq \tau(\hat{J},b_k)$, and, at the same time, have a positive lag at time b_k (as provided by (26)), τ_g must belong to $\tau^{MPS}(\hat{J},b_k)$, because $\tau^{DPS}(\hat{J},b_k)$ contains only tasks with a non-positive lag. In particular, by Definition 3, this implies that τ_g has at least one pending job at time b_k. Defining as t_g the minimum time instant such that $[t_g, b_k]$ is a maximal time interval during which τ_g has always pending jobs in the MPS, from the previous property we have that this time instant exists (as a degenerate case, $t_g = b_k$). We prove item 2.a using t_g. In particular, we prove, first, that at all times t in the intersection $[t_g, b_k) \cap [\tilde{b}_q, b_k)$, the task τ_g is being served in the MPS and belongs to $\tau^{MPS}(\hat{J},t)$. Then we prove that $t_g < \tilde{b}_q$, and that this inequality plus the previous property imply item 2.a.

As for the first step, from the fact that τ_g belongs to $\tau^{MPS}(\hat{J},b_k)$, i.e., owns at least one pending job with a deadline earlier than or equal to d_i^j in the MPS at time b_k, and

[8] Multiple non-growing-Lag intervals for different job portions may start at time 0, hence $b_k = 0$ may hold even for $k > 1$.

the fact that, by definition of t_g, τ_g has always pending jobs in the MPS throughout $[t_g, b_k]$, it follows that τ_g has always at least one pending job with a deadline earlier than or equal to d_i^j in the MPS throughout $[t_g, b_k]$ (because the jobs of any task are both released and executed in increasing-deadline order). But, still by Definition 3, this implies that τ_g belongs to $\tau^{MPS}(\hat{J}, t)$ at all times t in $[t_g, b_k]$. In this respect, the intersection $[t_g, b_k] \cap [\tilde{b}_q, b_k)$ is a growing-Lag interval by definition of $\tilde{b}_q$. Therefore, by Lemma 16, $\tau^{MPS}(\hat{J}, t) \leq \lceil U_{sum} \rceil - 1 < M$ at all times t in $[t_g, b_k] \cap [\tilde{b}_q, b_k)$. This relation and the fact that τ_g belongs to $\tau^{MPS}(\hat{J}, t)$ at all times t in $[t_g, b_k]$ imply that τ_g is continuously served during $[t_g, b_k] \cap [\tilde{b}_q, b_k)$ by Lemma 9.

As for the second step, we consider first that, if $t_g < \tilde{b}_q$, then there exists a time instant s_g such that $t_g \leq s_g \leq \tilde{b}_q$ and τ_g is constantly in service during $[s_g, b_k)$. In fact, as a degenerate case, $s_g = \tilde{b}_q$, because from the previous step and $t_g < \tilde{b}_q$ we have that τ_g is constantly in service at least during $[\tilde{b}_q, b_k)$. We prove then that s_g exists by proving, by contradiction, that $t_g < \tilde{b}_q$. Suppose that $t_g \geq \tilde{b}_q$ holds. By definition of t_g, τ_g starts to have pending jobs at time t_g. As a consequence, according to what we proved in the previous step, τ_g enters $\tau^{MPS}(\hat{J}, t)$ at time t_g, and therefore $\text{lag}_g(t_g) = 0$ holds by Lemma 15. This finally implies $\text{lag}_g(b_k) \leq 0$ by Lemma 12 and because τ_g is constantly served, and continuously belongs to $\tau^{MPS}(\hat{J}, t)$, during $[t_g, b_k)$. But this is absurd by (26). In the end, the above-defined time instant s_g exists.

The fact that τ_g is constantly served during $[s_g, b_k)$ also implies that s_g has pending jobs also throughout $[s_g, \tilde{b}_q)$. In this respect, the jobs of τ_g executed before time $\tilde{b}_q$ have of course a deadline earlier than or equal to that of the job of τ_g in execution at time $\tilde{b}_q$. As a consequence, since τ_g belongs to $\tau^{MPS}(\hat{J}, \tilde{b}_q)$, then τ_g continuously belongs to $\tau^{MPS}(\hat{J}, t)$ also during $[s_g, \tilde{b}_q)$. This completes the proof of item 2.a.

We prove now item 2.b. Using the same arguments as above, we also have, again by Lemma 12 and since τ_g is continuously served during $[s_g, b_k)$, that for τ_g to have a strictly positive lag at time b_k, the lag of the task must be strictly positive at time s_g. For this to happen, by Lemma 14, $s_g > r_g^l$ must hold. In this respect, by definition of a head, the only possibility for the head $\hat{J}_g$ to start after time r_g^l is that the head is blocked by priority.

As for item 2.c, from item 2.a we have that every task τ_g of $\hat{\tau}$ constantly belongs to $\tau^{MPS}(\hat{J}, t)$ during $[s_g, b_k)$, and hence during $[\tilde{b}_q, b_k)$. But, by Lemma 16, $|\hat{\tau}| \leq \lceil U_{sum} \rceil - 1$ at all times during $[\tilde{b}_q, b_k)$ (because $[\tilde{b}_q, b_k)$ is a growing-Lag interval).

We can finally prove also the sub-case $k = 1$ of item 1. If $k = 1$, then the k-th non-growing-Lag interval is the very first non-growing-Lag interval. As a consequence, by Lemma 19, no job may have been blocked by priority before time b_k. Thanks to item 2.b, $\hat{\tau}$ is therefore the empty set. □

In Figure 5 we show an example of the chains of the tasks in $\hat{\tau}$ in execution at the beginning $\tilde{b}_q$ of the growing-Lag interval that precedes the k-th non-growing-Lag interval $[b_k, f_k)$. The MPS has 7 processors. Rectangles represent job portions executed on the processors as in Figure 1.B. We assume that the non-growing-Lag interval $[b_k, f_k)$ shown in Figure 5 is the last non-growing-Lag interval for a portion $\hat{J} \subseteq J_i^j$ blocked by priority. The portion $\hat{J}$ is depicted as a filled, dark rectangle. Some rectangles are drawn with dashed lines during $[\tilde{b}_q, b_k)$, meaning that the portions they represent may or may not be executed during $[\tilde{b}_q, b_k)$. In fact, by Lemma 16, what

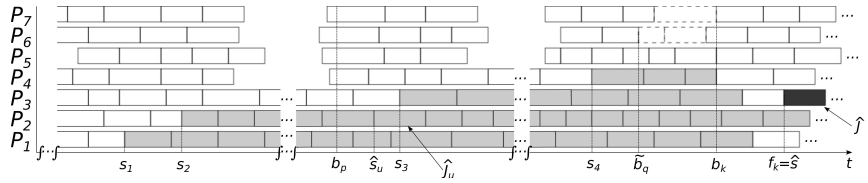


Fig. 5: Chains of tasks in $\hat{\tau}$. Rectangles represent job portions executed on processors. In particular, the dark rectangle represents the execution of the portion $\hat{f}$ blocked by priority, whose last non-growing-Lag period is $[b_k, f_k)$, whereas filled light rectangles represent the chains, of the tasks in $\hat{\tau}$, in execution at the beginning $\tilde{b}_q$ of the growing-Lag interval that precedes $[b_k, f_k)$.

matters for $[\tilde{b}_q, b_k)$ to be a growing-Lag period, is only that these portions have a strictly later deadline than d_i^j. In the figure we assume that $|\hat{\tau}| = 4$, and we represent the chains, in execution at time $\tilde{b}_q$, of the four tasks in $\hat{\tau}$ with filled, light rectangles. Note that the figure shows the possible case where a chain happens to terminate exactly at time b_k (on P_4). The figure also shows other information that we use later.

We introduce now some abbreviations and assumptions that we use in the rest of this paper to simplify the proofs. With the following assumptions we do not lose generality, because we just change the values of the indexes of the tasks in $\hat{\tau}$.

Assumption 1 *For each task $\tau_g \in \hat{\tau}$, hereafter we refer to its chain in execution at time $\tilde{b}_q$ (with $\tilde{b}_q$ defined as in Lemma 20) as just* its chain*, and we denote by s_g the start time of the chain. Second, we suppose that $\hat{\tau} = \{\tau_1, \tau_2, \ldots, \tau_G\}$, and $s_1 \leq s_2 \leq \ldots \leq s_G$ holds, i.e., that the indexes of the tasks in $\hat{\tau}$ are ordered by the start times of the chains. In particular, for two consecutive tasks τ_v and τ_{v+1}, both belonging to $\hat{\tau}$ and such that their chain heads both start at the same time $s_v = s_{v+1}$, we assume that the chain head of τ_v has a deadline earlier than or equal to that of the chain head of τ_{v+1}. Finally, we also assume that the chain starting at time s_g, with $1 \leq g \leq G$, is executed on the g-th processor.*

By Assumption 1 and item 2.c in Lemma 20, we have

$$G = |\hat{\tau}| \leq \lceil U_{sum} \rceil - 1. \tag{58}$$

In Figure 5 the start times of the chains and the allocation of the chains on the processors do reflect Assumption 1. Note also that each chain head starts in a time interval during which all the processors are busy. This happens because, by item 2.b of Lemma 20, the heads of the chains are all blocked by priority.

Basing on Assumption 1, we can prove the following general property on the sum of the variations of the lags of the tasks in $\hat{\tau}$. This property comes in handy in a number of proofs.

Lemma 21 *Given an integer g with $2 \leq g \leq G$, and $g-1$ time instants $t_1, t_2, \ldots, t_{g-1}$, with $t_v \in [s_v, b_k)$, the following relation holds:*

$$\forall\, \bar{t} \in (\max_{1 \leq v \leq g-1} t_v, b_k) \quad \sum_{v=1}^{g-1} lag_v(t_v, \bar{t}) \leq 0. \tag{59}$$

Proof From $t_v \in [s_v, b_k)$ for all $v \in \{1,2,\ldots,g-1\}$, we have that, for all $\bar{t} \in (\max_{1\leq v\leq g-1} t_v, b_k)$ and all $v \in \{1,2,\ldots,g-1\}$, the following inclusions hold: $[t_v,\bar{t}) \subseteq [t_v,b_k) \subseteq [s_v,b_k)$. As for the time interval $[s_v,b_k)$, we have that τ_v is continuously served and continuously belongs to $\tau^{MPS}(\hat{J},t)$ during $[s_v,b_k)$, by item 2.a of Lemma 20. From this fact and $[t_v,\bar{t}) \subseteq [s_v,b_k)$, we have that $\text{lag}_v(t_v,\bar{t}) \stackrel{(1)}{=} \text{lag}_v(\bar{t}) - \text{lag}_v(t_v) \leq 0$ by Lemma 12. Replacing the last inequality in the LHS of the inequality in (59), we get the thesis. □

As we already said in the description of Step 0, we use the time-balance property stated in Lemma 6 to compute a lag-balance property that, in its turn, allows us to get a tighter upper bound to the total lag. The following lemma reports both the subset Υ_g that we use to obtain the lag-balance property from the time-balance property, and a time instant t_g^{max} that allows us to use Lemma 6 with this subset, i.e., such that the hypotheses of Lemma 6 hold with $\Upsilon = \Upsilon_g$ after setting $t_1 = t_g^{max}$. Since it may not be easy to visualize the job portions and the time instants considered in the lemma and in its proof, we show an example before proving the lemma.

Lemma 22 *Given a task $\tau_g \in \hat{\tau}$ with $g > 1$, and defining: $\hat{J}_g \subseteq J_g^l$ as the head of the chain of τ_g, $\Upsilon_g \subseteq \{\tau_1,\tau_2,\ldots,\tau_{g-1}\}$ as any subset of the first $g-1$ tasks in $\hat{\tau}$, $\hat{b}_v$ as the beginning, for each task $\tau_v \in \Upsilon_g \cup \{\tau_g\}$, of the busy interval of the job portion of τ_v in execution at time s_g, and $t_g^{max} \equiv \max_{\tau_v\in\Upsilon_g\cup\{\tau_g\}} \hat{b}_v$, we have that: (1) $t_g^{max} \in [\hat{b}_g, \min\{s_g,d_g^l\})$, and (2) $\Upsilon_g \subseteq \tau^{MPS}(\hat{J}_g,t_g^{max})$.*

Suppose that the chains of the tasks in $\hat{\tau}$ in Lemma 22 are those shown in Figure 5, and that $g = 3$ and $\Upsilon_3 = \{\tau_1,\tau_2\}$. With these assumptions, the job portions of the tasks in Υ_3 in execution at time $s_g = s_3$ are the ones in execution on processors $P1$ and $P2$ at time s_3.

Proof We start by proving item 1, i.e., that $\hat{b}_g \leq t_g^{max}$ and $t_g^{max} < \min\{s_g,d_g^l\}$ hold. The first inequality trivially holds by definition of t_g^{max}. As for the second inequality, we prove first that $t_g^{max} < s_g$ and then that $t_g^{max} < d_g^l$. Regarding $t_g^{max} < s_g$, since the heads of the chains of the tasks in $\hat{\tau}$ are all blocked by priority, we have that $\hat{b}_v < \hat{s}_v$ for all τ_v in $\Upsilon_g \cup \{\tau_g\}$. Therefore, by definition of t_g^{max}, $t_g^{max} = \max_{\tau_v\in\Upsilon_g\cup\{\tau_g\}} \hat{b}_v < \max_{\tau_v\in\Upsilon_g\cup\{\tau_g\}} \hat{s}_v = s_g$, where the last equality follows from Assumption 1.

Given a task $\tau_v \in \Upsilon_g$, let $\hat{J}_u$ be the portion of the task in execution at time s_g, and let $\hat{s}_u$ and $\hat{d}_u$ be the start time and the deadline of $\hat{J}_u$. Figure 5 shows both $\hat{J}_u$ and $\hat{s}_u$ assuming $g = 3$ and $v = 2$. Since $\hat{J}_g$ is blocked by priority at least right before time s_g (by item 2.b of Lemma 20), but $\hat{J}_u$ is already in execution by time s_g (time s_3 in Figure 5), we have that $\hat{d}_u \leq d_g^l$ must hold, by Lemma 1, for $\hat{J}_g$ not to preempt $\hat{J}_u$ before time s_g. Now let $\hat{J}_v$ be the head of the chain $\hat{J}_u$ belongs to, namely the portion of τ_2 starting at time s_2 in Figure 5. If $\hat{d}_v$ is the deadline of $\hat{J}_v$, then we also have that $\hat{d}_v \leq \hat{d}_u$ (because either $\hat{J}_u = \hat{J}_v$ or $\hat{J}_v$ belongs to a previous job, of τ_v, than that $\hat{J}_u$ belongs to). Combining this inequality with $\hat{d}_u \leq d_g^l$, we get $\hat{d}_v \leq d_g^l$.

Since the last inequality holds for any task $\tau_v \in \Upsilon_g$, it follows that, denoting by $\hat{d}_v$ the deadline of the head of the chain of each task $\tau_v \in \Upsilon_g$ in execution at time s_g, and

defining $\hat{d}_g \equiv d_g^l$, we have $\forall \tau_v \in \Upsilon_g \cup \{\tau_g\}$ $\hat{d}_v \leq d_g^l$. Considering also Lemma 8, this implies that $\forall \tau_v \in \Upsilon_g \cup \{\tau_g\}$, $\hat{b}_v < \hat{d}_v \leq d_g^l$. As a consequence, by definition of t_g^{max}, $t_g^{max} = \max_{\tau_v \in \Upsilon_g \cup \{\tau_g\}} \hat{b}_v < d_g^l$.

We prove now item 2, i.e., that every task τ_v in Υ_g belongs to $\tau^{MPS}(\hat{J}_g, t_g^{max})$. Consider the chain of a task $\tau_v \in \Upsilon_g$ in execution at time s_g and the deadline $\hat{d}_u$ of the above-defined portion $\hat{J}_u$ in execution at time s_g. As shown in Figure 5 for $g = 3$ and $v = 2$, the set of the portions of the chain of τ_v executed during $[s_v, s_g]$ ($[s_2, s_3]$ in the figure) consists of $\hat{J}_u$, plus possible other portions executed before $\hat{J}_u$ (if $\hat{J}_u$ is not the head of the chain). The latter portions therefore belong to previous jobs of τ_v, with respect to the job $\hat{J}_u$ belongs to. As a consequence, since we already proved that $\hat{d}_u \leq d_g^l$, all these portions have a deadline earlier than or equal to d_g^l.

Thanks to the above property, and by definition of $\tau^{MPS}(\hat{J}_g, t)$ (Definition 3), to prove item 2 we prove that at least one of these portions is pending in the MPS at time t_g^{max}. To this purpose, suppose first that $t_g^{max} < s_v$ (i.e., $t_g^{max} < s_2$ in Figure 5). In this case we can consider that, by definition, $t_g^{max} \geq \hat{b}_v$, and, by Lemma 8, $\hat{b}_v$ is at least equal to the release time of the job $\hat{J}_v$ belongs to. As a consequence, the head $\hat{J}_v$ is already pending (although still blocked by priority) at time t_g^{max}. As for the other alternative, namely $t_g^{max} \geq s_v$, we already proved, at the beginning of this proof, that $t_g^{max} < s_g$ also holds. In the end, $t_g^{max} \in [s_v, s_g)$ (namely, $t_g^{max} \in [s_2, s_3)$ in Figure 5), i.e., one of the portions of τ_v executed during $[s_v, s_g]$ is executing, and thus pending (according to our definition of *pending*), at time t_g^{max}. Thus, in both alternatives, a portion with a deadline earlier than or equal to d_g^l is pending at time t_g^{max}. □

To further simplify proofs, we make also the following last assumption in the lemmas by which we compute an upper bound to the total lag.

Assumption 2 *Given any job portion $\hat{J} \subseteq J_i^j$ blocked by priority, and: (a) assuming that the k-th non-growing Lag interval is the last non-growing Lag interval for that portion, (b) making Assumption 1, and (c) denoting, for every task τ_g in $\hat{\tau}$, by J_g^l the job to which the head of the chain of τ_g starting at time s_g belongs, we assume that $l_g^l = L_g$ holds.*

This assumption does not affect the correctness of the proof of the harmonic bound. When needed, we use the following lemma to prove this fact.

Lemma 23 *Suppose that the k-th non-growing-Lag interval is a non-growing-Lag interval for a job portion $\hat{J} \subseteq J_i^j$ blocked by priority, and consider any task $\tau_g \in \hat{\tau}$. Let $\hat{J}_g \in J_g^l$ be the head of the chain of τ_g starting at time s_g. If the length of J_g^l is strictly lower than the maximum possible job length for the task, i.e., $l_g^j < L_g$, then the start time $\hat{s}$ of the portion $\hat{J}$ does not decrease if the job J_g^l is replaced with a job with a length equal to L_g (without changing any other parameter).*

Proof Let J_g be the long job with which the shorter job J_g^l is replaced. For brevity, we call *true scenario* the original scenario, and *artificial scenario* the one in which J_g^l is replaced with J_g. Since this replacement does not change either the deadline of any job, or the length of any job but J_g^l, it follows that, by Lemma 1, the schedule of all the jobs in the MPS during $[0, s_g]$ is exactly the same in both scenarios.

Consider then the time interval $[s_g, b_k)$. By item 2.a of Lemma 20 (and as shown in Figure 5 for every task in $\{\tau_1, \tau_2, \tau_3, \tau_4\}$), in both scenarios the g-th processor is constantly busy serving task τ_g during $[s_g, b_k)$. In addition, in both scenarios, all the parameters of all the jobs of all the tasks but τ_g are the same. Thus, during $[s_g, b_k)$, the schedule of all the jobs of all the tasks but τ_g is exactly the same in both scenarios. In particular, this implies that the value of b_k is the same in both scenarios, because, by Lemma 16, the k-th non-growing-Lag period starts when at least a new task enters $\tau^{MPS}(\hat{J}, t)$, and τ_g already belongs to $\tau^{MPS}(\hat{J}, b_k)$ by item 2.a of Lemma 20.

As a last step, we analyze what happens during $[b_k, \hat{s}]$. To this purpose, we define, firstly, as f_g the time instant at which the chain of τ_g terminates. Since the parameters of all the jobs but J_g^l are the same in both scenarios, whereas the head $\hat{J}_g$ is longer in the artificial scenario (because J_g is longer than J_g^l and the schedule of the jobs of τ_g is identical in both scenarios up to time s_g), it follows that the value of f_g in the artificial scenario cannot be lower than in the true scenario. Consider for example the chain of τ_1 in Figure 5, and visualize what happens after increasing the size of the portion of τ_1 that starts at time s_1.

We consider now two alternatives for the true scenario. The first is that $f_g \geq \hat{s}$. This is what happens, e.g., with the chain of τ_2 in Figure 5. As highlighted by the figure, since $\hat{J}$ is blocked by priority, and hence τ_i cannot be in service right before time $\hat{s}$, $\tau_g \neq \tau_i$ holds. Moreover, recall that, in the artificial scenario, f_g cannot be lower than in the true scenario. Finally, the parameters of all the jobs but J_g^l are the same in both scenario. In the end, during $[b_k, \hat{s}]$ the schedule of all the job portions of all the tasks but τ_g is exactly the same in both scenarios. Therefore $\hat{s}$ has the same value in both scenarios.

Finally, consider the second alternative, i.e., $f_g < \hat{s}$. This is what happens in Figure 5 for all the chains except for that of τ_2. According to the above arguments, the amount of service received by every task by time b_k is exactly the same in both scenarios. In addition, in the artificial scenario the value of f_g cannot be lower than in the true scenario. Considering also that the parameters of all the jobs but J_g^l, which becomes longer, are the same in both scenarios, it follows that, in the artificial scenario, the MPS cannot have less work to do during $[b_k, \hat{s}]$ than in the true scenario. To visualize this fact, suppose that $\tau_g = \tau_1$ in Figure 5, and imagine that, in the artificial scenario, the chain of τ_1 terminates at the same time as or later than in the true scenario. In the end, since the MPS works at constant total speed M during $[b_k, \hat{s}]$ in both scenarios, in the artificial scenario $\hat{s}$ cannot be lower than in the true scenario. □

8 Proof of the harmonic bound

Using the machinery provided in Section 7, in this section we prove Theorem 1 by the steps outlined in Section 5 (and shown in Figure 3).

8.1 Step 0: prove the time-balance property

Step 0 consists in proving Lemma 6 (Section 5). For the reader's convenience, we sum up the statement of the lemma before providing the proof. Given: (a) the busy

interval $[\hat{b},\hat{s})$ for a portion $\hat{f} \subseteq J_i^j$, (b) any time instant $t_1 \in [\hat{b}, \min\{d_i^j, \hat{s}\})$, and (c) any subset $\Upsilon \subseteq \tau^{MPS}(\hat{f},t_1)$ not containing τ_i, including the empty set, we have

$$\hat{s} - d_i^j \leq \frac{\sum_{h\in\tau(\hat{f},t_1)} \mathrm{lag}_h(t_1) - \sum_{v\in\Upsilon} \mathrm{lag}_v(\hat{s}) - \hat{l}}{M - \sum_{v\in\Upsilon} U_v}, \tag{60}$$

where $\hat{l}$ is the size, in service units, of the part of the job J_i^j not yet executed in the MPS by time $\hat{s}$.

Proof (Lemma 6) We start from (35) and split both the work done by the MPS and the work done by DPS into two components: total service provided to the tasks in $\tau \setminus \Upsilon$, and total service provided to the tasks in Υ. This split helps us let the time-balance property emerge. We have then the following relations, where the inclusion on the last equality follows from $\Upsilon \subseteq \tau^{MPS}(\hat{f},t_1) \subseteq \tau(\hat{f},t_1)$,

$$\begin{aligned}
\hat{s} - d_i^j &\overset{(35)}{\leq} \frac{\sum_{h\in\tau} W_h^{MPS}(t_1,\hat{s}) - \sum_{h\in\tau} W_h^{DPS}(t_1,d_i^j)}{M} && \overset{\text{splitting}}{=} \\
&\frac{\sum_{h\in\tau\setminus\Upsilon} W_h^{MPS}(t_1,\hat{s}) + \sum_{v\in\Upsilon} W_v^{MPS}(t_1,\hat{s}) - \sum_{h\in\tau\setminus\Upsilon} W_v^{DPS}(t_1,d_i^j) - \sum_{v\in\Upsilon} W_v^{DPS}(t_1,d_i^j)}{M} && \overset{\text{moving terms}}{=} \\
&\frac{\left(\sum_{h\in\tau\setminus\Upsilon} W_h^{MPS}(t_1,\hat{s}) - \sum_{h\in\tau\setminus\Upsilon} W_v^{DPS}(t_1,d_i^j)\right) + \sum_{v\in\Upsilon} W_v^{MPS}(t_1,\hat{s}) - \sum_{v\in\Upsilon} W_v^{DPS}(t_1,d_i^j)}{M} && \overset{(49)}{\leq} \\
&\frac{\left(\sum_{h\in\tau(\hat{f},t_1)\setminus\Upsilon} \mathrm{lag}_h(t_1) - \hat{l}\right) + \sum_{v\in\Upsilon} W_v^{MPS}(t_1,\hat{s}) - \sum_{v\in\Upsilon} W_v^{DPS}(t_1,d_i^j)}{M} && \overset{\Upsilon\subseteq\tau(\hat{f},t_1)}{=} \\
&\frac{\left(\sum_{h\in\tau(\hat{f},t_1)} \mathrm{lag}_h(t_1) - \sum_{v\in\Upsilon} \mathrm{lag}_v(t_1) - \hat{l}\right) + \sum_{v\in\Upsilon} W_v^{MPS}(t_1,\hat{s}) - \sum_{v\in\Upsilon} W_v^{DPS}(t_1,d_i^j)}{M}.
\end{aligned} \tag{61}$$

The last two terms in the numerator of the last line of (61) enable us to let the time-balance property come into play. To this purpose, first we rewrite these terms as follows:

$$\begin{aligned}
&\sum_{v\in\Upsilon} W_v^{MPS}(t_1,\hat{s}) - \sum_{v\in\Upsilon} W_v^{DPS}(t_1,d_i^j) && \overset{(2)}{=} \\
&\sum_{v\in\Upsilon} W_v^{MPS}(t_1,\hat{s}) - \sum_{v\in\Upsilon} \left[W_v^{DPS}(t_1,\hat{s}) + W_v^{DPS}(\hat{s},d_i^j)\right] && \overset{\text{rearranging}}{=} \\
&- \left\{\sum_{v\in\Upsilon} [W_v^{DPS}(t_1,\hat{s}) - W_v^{MPS}(t_1,\hat{s})]\right\} - \sum_{v\in\Upsilon} W_v^{DPS}(\hat{s},d_i^j).
\end{aligned} \tag{62}$$

Now, from $\Upsilon \subseteq \tau^{MPS}(\hat{f},t_1)$ it follows that the MPS can have executed, by time t_1 and for any task in Υ, only jobs with a deadline earlier than or equal to d_i^j. The same then holds also by time $\hat{s}$, because, by Definition 2, the MPS can execute only jobs with a deadline earlier than or equal to d_i^j during $[t_1,\hat{s})$. As a consequence, by Lemma 11,

$$\begin{aligned}
&\forall \tau_v \in \Upsilon \quad \mathrm{lag}_v(t_1,\hat{s}) = \left[W_v^{DPS}(\hat{s}) - W_v^{MPS}(\hat{s})\right] - \left[W_v^{DPS}(t_1) - W_v^{MPS}(t_1)\right] = \\
&\left[W_v^{DPS}(\hat{s}) - W_v^{DPS}(t_1)\right] - \left[W_v^{MPS}(\hat{s}) - W_v^{MPS}(t_1)\right] \overset{(1)}{=} W_v^{DPS}(t_1,\hat{s}) - W_v^{MPS}(t_1,\hat{s}).
\end{aligned} \tag{63}$$

Replacing (63) in the last line of (62), we get

$$\begin{aligned}
&\sum_{v\in\Upsilon} W_v^{MPS}(t_1,\hat{s}) - \sum_{v\in\Upsilon} W_v^{DPS}(t_1,d_i^j) && = \\
&-\sum_{v\in\Upsilon} \mathrm{lag}_v(t_1,\hat{s}) + \sum_{v\in\Upsilon} W_v^{DPS}(d_i^j,\hat{s}) && \overset{\text{Def. 4}}{\leq} \\
&-\sum_{v\in\Upsilon} \mathrm{lag}_v(t_1,\hat{s}) + (s - d_i^j) \sum_{v\in\Upsilon} U_v.
\end{aligned} \tag{64}$$

Finally, replacing (64) in the last line of (61), we have

$$\begin{aligned}
\hat{s}-d_i^j \leq \frac{\left(\sum_{h\in\tau(\hat{f},t_1)}\text{lag}_h(t_1)-\sum_{v\in\Upsilon}\text{lag}_v(t_1)-\hat{l}\right)-\sum_{v\in\Upsilon}\text{lag}_v(t_1,\hat{s})+(s-d_i^j)\sum_{v\in\Upsilon}U_v}{M} &\overset{\text{rearranging}}{\iff}\\
(\hat{s}-d_i^j)\frac{M-\sum_{v\in\Upsilon}U_v}{M} \leq \frac{\sum_{h\in\tau(\hat{f},t_1)}\text{lag}_h(t_1)-\sum_{v\in\Upsilon}\text{lag}_v(t_1)-\sum_{v\in\Upsilon}\text{lag}_v(t_1,\hat{s})-\hat{l}}{M} &\overset{(3)}{\iff}\\
(\hat{s}-d_i^j)\frac{M-\sum_{v\in\Upsilon}U_v}{M} \leq \frac{\sum_{h\in\tau(\hat{f},t_1)}\text{lag}_h(t_1)-\sum_{v\in\Upsilon}\text{lag}_v(\hat{s})-\hat{l}}{M} &\overset{\text{solving for } \hat{s}-d_i^j}{\iff}\\
\hat{s}-d_i^j \leq \frac{\sum_{h\in\tau(\hat{f},t_1)}\text{lag}_h(t_1)-\sum_{v\in\Upsilon}\text{lag}_v(\hat{s})-\hat{l}}{M-\sum_{v\in\Upsilon}U_v}.&
\end{aligned} \tag{65}$$

□

8.2 Step 1: compute an upper bound to the tardiness as a function of the total lag

Step 1 consists in proving Lemma 4. For the reader's convenience, we sum up also the statement of Lemma 4 before giving the proof. For every job J_i^j, there exists at least one integer k for which the following inequality holds:

$$f_i^j - d_i^j \leq \frac{\Lambda(k)}{M} + \frac{M-1}{M}C_i. \tag{66}$$

Proof (Lemma 4) By Corollary 1, if none of the following two conditions holds, then $f_i^j - d_i^j \leq 0$, and hence (66) trivially holds: (a) the last portion of J_i^j is blocked by priority, or (b) J_i^j is wholly executed without interruption after being blocked by precedence right before starting, and the head of the chain J_i^j belongs to is in its turn blocked by priority. We complete then the proof by demonstrating that (66) holds also if any of the these two conditions holds.

About the first condition, suppose that: $\hat{J} \subseteq J_i^j$ is the last portion of J_i^j, $[\hat{b},\hat{s})$ is the busy interval of $\hat{J}$, and $[\hat{b},\hat{s})$ belongs to the k-th non-growing-Lag interval $[b_k,\hat{s})$. Let $\hat{c}$ and $\hat{l}$ denote, respectively, the time needed to execute $\hat{J}$ on a unit-speed processor and the length of $\hat{J}$ in service units. It follows that $\hat{c}$ is numerically equal to $\hat{l}$, but it is measured in time units instead of service units. With some abuse of notation, we can write, concisely, that $\frac{\hat{l}}{M} = \frac{\hat{c}}{M}$, assuming that the denominator is a speed in the first fraction and a pure number in the second. Using this equality, plus Lemma 6 with $\Upsilon = \emptyset$ and $t_1 = \hat{b}$, we get

$$\hat{s} - d_i^j \overset{\text{Lemma 6}}{\leq} \frac{\sum_{h\in\tau(\hat{f},\hat{b})}\text{lag}_h(\hat{b})-\hat{l}}{M}. \tag{67}$$

Using this inequality, we can write:

$$\begin{aligned}
f_i^j - d_i^j = \hat{s}+\hat{c}-d_i^j = \hat{s}-d_i^j+\hat{c} \overset{(67)}{\leq} \frac{\sum_{h\in\tau(\hat{f},\hat{b})}\text{lag}_h(\hat{b})-\hat{l}}{M}+\hat{c} &\overset{\substack{(57)\text{ with}\\ t_1=\hat{b}}}{\leq}\\
\frac{\sum_{h\in\tau(\hat{f},b_k)}\text{lag}_h(b_k)-\hat{l}}{M}+\hat{c} \overset{(25)}{\leq} \frac{\Lambda(k)-\hat{l}}{M}+\hat{c} = \frac{\Lambda(k)}{M}-\frac{\hat{l}}{M}+\hat{c} &= \\
\frac{\Lambda(k)}{M}-\frac{\hat{c}}{M}+\hat{c} = \frac{\Lambda(k)}{M}+\frac{M-1}{M}\hat{c} \leq \frac{\Lambda(k)}{M}+\frac{M-1}{M}C_i.&
\end{aligned} \tag{68}$$

Instead, if the second condition holds, let J_i^e, with $e < j$, be the job the head of the chain belongs to. We denote then as f_i^e and d_i^e the completion time and the

deadline of J_i^e. Since the head is blocked by priority and, according to the final note in Section 2.2, is the last portion of the job J_i^e, the difference $f_i^e - d_i^e$ can be upper-bounded by repeating the same steps (and hence obtaining the same upper bound) as in (68). The thesis then follows from the fact that, by Lemma 3, $f_i^j - d_i^j \leq f_i^e - d_i^e$. □

8.3 Step 2: upper-bound the total lag

Step 2 consists in proving Lemma 5. First we describe the strategy and the relations by which we prove the lemma, and then we report the main sub-steps by which we get to prove the lemma.

We prove Lemma 5 by induction. Denoting, for brevity, by B the RHS of (33), Lemma 5 states that $\Lambda(k) \leq B$ holds for all k. In this respect, as we show in Section 8.7, $\Lambda(1) \leq B$ is trivial to prove for any $B \geq 0$. Instead, regarding the inductive step, consider that, denoting by $\hat{J}_k$ the portion for which the total lag at time b_k is maximum, and by $\hat{d}_k$ the deadline of $\hat{J}_k$, we have

$$\begin{aligned} \Lambda(k) &\overset{(25)}{=} \max_{1\leq p\leq k} \lambda(p) = \max\left[\left(\max_{1\leq p\leq k-1} \lambda(p)\right), \lambda(k)\right] \overset{(25)}{=} \\ \max\left[\Lambda(k-1), \lambda(k)\right] &\overset{(24)}{=} \max\left[\Lambda(k-1), \sum_{h\in\tau(\hat{J}_k,b_k)} \mathrm{lag}_h\langle\hat{d}_k\rangle(b_k)\right]. \end{aligned} \tag{69}$$

By (69), the following implication holds:

$$\left(\Lambda(k-1) \leq B \implies \sum_{h\in\tau(\hat{J}_k,b_k)} \mathrm{lag}_h\langle\hat{d}_k\rangle(b_k) \leq B\right) \overset{(69)}{\Longrightarrow} \left(\Lambda(k-1) \leq B \Rightarrow \Lambda(k) \leq B\right). \tag{70}$$

The consequent in (70) is exactly what has to be proved, as an inductive step, to prove by induction that $\Lambda(k) \leq B$ holds for all k. As a consequence, we can prove that $\Lambda(k) \leq B$ holds for all k, by proving, as a base case, that $\Lambda(1) \leq B$, and, for the inductive step, the antecedent in (70), i.e., the implication

$$\Lambda(k-1) \leq B \implies \sum_{h\in\tau(\hat{J}_k,b_k)} \mathrm{lag}_h\langle\hat{d}_k\rangle(b_k) \leq B. \tag{71}$$

To prove the implication (71), which holds for the job portion $\hat{J}_k$ for which the total lag at time b_k is maximum, we prove that the same implication holds for any job portion $\hat{J} \subseteq J_i^j$, blocked by priority and for which the k-th non-growing-Lag interval is the last non-growing-Lag interval. In particular, we use the following relations to prove that, if the antecedent in (71) holds, then the consequent in (71) holds for the

above generic portion $\hat{J} \subseteq J_i^j$:

$$\sum_{h \in \tau(\hat{J},b_k)} \text{lag}_h\langle d_i^j \rangle(b_k) \overset{(23)}{=} \sum_{h \in \tau(\hat{J},b_k)} \text{lag}_h(b_k) \overset{(27)}{\leq} \sum_{g \in \hat{\tau}} \text{lag}_g(b_k) \overset{(3)}{=} \sum_{g \in \hat{\tau}} \text{lag}_g(s_g) + \sum_{g \in \hat{\tau}} \text{lag}_g(s_g, b_k). \tag{72}$$

where $\text{lag}_g(s_g, b_k) = \text{lag}_g(b_k) - \text{lag}_g(s_g)$. In view of (72), we prove that $\sum_{h \in \tau(\hat{J},b_k)} \text{lag}_h(b_k) \leq B$ holds by, first, computing an upper bound to $\sum_{g \in \hat{\tau}} \text{lag}_g(s_g)$, and then adding $\sum_{g \in \hat{\tau}} \text{lag}_g(s_g, b_k)$ to this bound. To get an upper bound to $\sum_{g \in \hat{\tau}} \text{lag}_g(s_g)$, we compute a bound to each lag in the summation, and then sum these per-task bounds. We can now state, even more precisely, that the peculiarity of our proofs lies in how we compute a bound to each lag in the sum $\sum_{g \in \hat{\tau}} \text{lag}_g(s_g)$.

We take the following four sub-steps to get to prove Lemma 5. The first three sub-steps are also shown in Figure 3, to further help understand their role in the proof. The bounds computed in the first three sub-steps are all in open form, in that they contain at least one term whose value has not yet been computed ($\Lambda(k-1)$). To simplify proofs, in all the next lemmas we assume that Assumption 2 holds, which implies, in particular, that the head of each task τ_g starting at time s_g belongs to a job of length L_g. The proofs of the lemmas are provided from Section 8.4 onwards.

Sub-step 2.1: Compute an upper bound in open form to $lag_g(s_g)$ for $\tau_g \in \hat{\tau}$

As a first sub-step, we prove the following lemma, stating the lag-balance property.

Lemma 24 (Lag-balance property) *If $\Upsilon_g \subseteq \{\tau_1, \tau_2, \ldots, \tau_{g-1}\}$ is any subset of the first $g-1$ tasks in $\hat{\tau}$, and $\hat{l}_g$ is the size, in service units, of the head $\hat{J}_g \subseteq J_g^l$ starting at time s_g, then the following inequality holds for the g-th task:*

$$lag_g(s_g) \leq U_g \frac{\Lambda(k-1) - \sum_{v \in \Upsilon_g} lag_v(s_g) - \hat{l}_g}{M - \sum_{v \in \Upsilon_g} U_v} + \hat{l}_g. \tag{73}$$

If $\Upsilon_g = \{\tau_1, \tau_2, \ldots, \tau_{g-1}\}$, then the bound (73) to $\text{lag}_g(s_g)$ decreases as the sum of the first $g-1$ addends in $\sum_{g \in \hat{\tau}} \text{lag}_g(s_g)$ increases (in this respect, recall Assumption 1 in Section 7.7 regarding the indexes of the tasks in $\hat{\tau}$). This internal counterbalance among the lags of the tasks in $\hat{\tau}$ is the key property that allows us to get a tighter bound to the whole sum $\sum_{g \in \hat{\tau}} \text{lag}_g(s_g)$, and hence to the total lag by (72).

Sub-step 2.2: Compute an upper bound to $\sum_{g \in \hat{\tau}} lag_g(s_g)$ in open form

In this sub-step, we compute an upper bound to $\sum_{g \in \hat{\tau}} \text{lag}_g(s_g)$ using Lemma 24. To simplify the notation, in this and in the next subsections we use the following definitions and properties:

$$M_g \equiv M - \sum_{v=1}^{g-1} U_v, \tag{74}$$

$$M_g - U_g = M - \sum_{v=1}^{g-1} U_v - U_g = M_{g+1}, \tag{75}$$

$$\forall \tau_g \in \hat{\tau} \;\; \Delta_g \equiv \sum_{v=1}^{g-1} \mathrm{lag}_v(s_v, s_g) \overset{\substack{(59) \text{ with} \\ t_v = s_v,\, \tilde{t} = s_g}}{\leq} 0. \tag{76}$$

Note that M_g is a special case of the more general quantity $M_g^{\tau'}$ in (4). In addition, recall that we denote as G the number of tasks in $\hat{\tau}$ (Assumption 1 in Section 7.7).

Lemma 25 *The following inequality holds:*

$$\sum_{g=1}^{G} lag_g(s_g) \leq M_{G+1} \cdot \left(\Lambda(k-1) \sum_{g=1}^{G} \frac{U_g}{M_g M_{g+1}} - \sum_{g=1}^{G} \frac{U_g \Delta_g}{M_g M_{g+1}} + \sum_{g=1}^{G} \frac{L_g}{M_g} \right). \tag{77}$$

Sub-step 2.3: Compute an upper bound to the total lag in open form

As a penultimate sub-step, we compute an upper bound to $\sum_{h \in \tau(\hat{J}, b_k)} \mathrm{lag}_h(b_k)$ in open form through (72), and, in particular, by adding $\sum_{g \in \hat{\tau}} \mathrm{lag}_g(s_g, b_k)$ to the bound (77).

Lemma 26 *For every job portion $\hat{J}$ for which $[b_k, f_k)$ is the last non-growing-Lag interval, we have*

$$\sum_{h \in \tau(\hat{J}, b_k)} lag_h(b_k) \leq M_{G+1} \cdot \left(\Lambda(k-1) \sum_{g=1}^{G} \frac{U_g}{M_g M_{g+1}} + \sum_{g=1}^{G} \frac{L_g}{M_g} \right). \tag{78}$$

Sub-step 2.4: Compute an upper bound to the total lag and to the tardiness in closed form

As a last sub-step, we prove first Lemma 5, i.e., that $\Lambda(k) \leq B$ holds for all k. Then we prove Theorem 1 by just replacing the bound $\Lambda(k) \leq B$ in (31).

8.4 Sub-step 2.1: Compute an upper bound in open form to $\mathrm{lag}_g(s_g)$ for $\tau_g \in \hat{\tau}$

For the reader's convenience, we repeat the statement of Lemma 24 here. If $\Upsilon_g \subseteq \{\tau_1, \tau_2, \ldots, \tau_{g-1}\}$ is any subset of the first $g-1$ tasks in $\hat{\tau}$, and $\hat{l}_g$ is the size, in service units, of the head $\hat{J}_g \subseteq J_g^l$ starting at time s_g, then the following inequality holds for the g-th task:

$$\mathrm{lag}_g(s_g) \leq U_g \frac{\Lambda(k-1) - \sum_{v \in \Upsilon_g} \mathrm{lag}_v(s_g) - \hat{l}_g}{M - \sum_{v \in \Upsilon_g} U_v} + \hat{l}_g. \tag{79}$$

Proof (Lemma 24) First, we derive a bound to $s_g - d_g^l$, and then turn this bound into (79). Defining $\hat{b}_g$ as the beginning of the busy interval of $\hat{J}_g$ and t_g^{max} as in Lemma 22, we have, by Lemma 22, that $t_g^{max} \in [\hat{b}_g, \min\{s_g, d_g^l\})$ and $\Upsilon_g \subseteq \tau^{MPS}(\hat{J}_g, t_g^{max})$. This implies that Lemma 6 holds for $\hat{J}_g$ after setting $\hat{f} = \hat{J}_g$, $J_i^j = J_g^l$, $\hat{b} = \hat{b}_g$, $\hat{s} = s_g$, $t_1 = t_g^{max}$, $\Upsilon = \Upsilon_g$ and $\hat{l} = \hat{l}_g$. We have then:

$$s_g - d_g^l \overset{\text{Lemmas 22 and 6}}{\leq} \frac{\sum_{h \in \tau(\hat{J}_g, t_g^{max})} \text{lag}_h\langle d_g^l \rangle(t_g^{max}) - \sum_{v \in \Upsilon_g} \text{lag}_v\langle d_g^l \rangle(s_g) - \hat{l}_g}{M - \sum_{v \in \Upsilon_g} U_v}, \tag{80}$$

where we used the general form (22) for the two lag functions. In fact, in both expressions the reference deadline is d_g^l, and not d_i^j as in (23).

As a first step to turn (80) into (79), we prove that the first sum of lags in the RHS of (80) is upper-bounded by $\Lambda(k-1)$. Let $[\hat{b}_g, s_g)$ and $[b_p, f_p)$ be, respectively, the busy interval and the last non-growing-Lag interval for $\hat{J}_g$. Since $t_g^{max} \in [\hat{b}_g, s_g)$, we have also that Lemma 19 holds for $\hat{J}_g$ and t_g^{max} after setting $\hat{f} = \hat{J}_g$, $\hat{b} = \hat{b}_g$, $\hat{s} = s_g$, $t_1 = t_g^{max}$ and $b_k = b_p$. Replacing these equalities in (57) and writing explicitly the first argument d_g^l in the lag functions, we get

$$\sum_{h \in \tau(\hat{J}_g, t_g^{max})} \text{lag}_h\langle d_g^l \rangle(t_g^{max}) \leq \sum_{h \in \tau(\hat{J}_g, b_p)} \text{lag}_h\langle d_g^l \rangle(b_p). \tag{81}$$

We prove now that $p \leq k-1$, from which we derive that the RHS of (81) is upper-bounded by $\Lambda(k-1)$. By item 2.a of Lemma 20, $s_g \leq \tilde{b}_q$ holds, where $\tilde{b}_q$ is the beginning of the growing-Lag interval that terminates at time b_k (see Figure 5). Since $\tilde{b}_q < b_k$, it follows that $s_g < b_k$ holds too. But, by Definition 2, $\hat{b}_g < s_g$, and, still by Lemma 19, $b_p \leq \hat{b}_g$. Combining the last three inequalities, we have $b_p < b_k$, which implies $p \leq k-1$ by the ordering among busy intervals (Definition 10). See for example Figure 5, showing a possible b_p in case $g = 3$. We have

$$\sum_{h \in \tau(\hat{J}_g, b_p)} \text{lag}_h\langle d_g^l \rangle(b_p) \overset{\text{(25), (24) and } p \leq k-1}{\leq} \Lambda(k-1). \tag{82}$$

As for the other sum, $\sum_{v \in \Upsilon_g} \text{lag}_v\langle d_g^l \rangle(s_g)$, recall that Υ_g is a subset of $\hat{\tau}$ by Assumption 1. Using the deadlines d_g^l and d_i^j of J_g^l and J_i^j (J_i^j is the reference job in the definition of $\hat{\tau}$), we define $m(v) \equiv \max_{d_v^n \leq d_g^l} n$ and $m'(v) \equiv \max_{d_v^n \leq d_i^j} n$, i.e., we define $m(v)$ and $m'(v)$ as the indexes of the latest-deadline jobs of τ_v, among those with a deadline earlier than or equal to, respectively, d_g^l and d_i^j. Since $d_g^l \leq d_i^j$ holds by Definition 3 and item 2.a of Lemma 20, it follows that $m(v) \leq m'(v)$ holds for all

v in Υ_g. Using this inequality, we can write

$$\begin{aligned}\sum_{v\in\Upsilon_g} \mathrm{lag}_v\langle d_g^l\rangle(s_g) &\overset{(22)+(21)}{=} \sum_{v\in\Upsilon_g}\left[W_v^{DPS}(s_g) - \min(W_v^{MPS}(s_g), \sum_{n=1}^{m(v)} l_v^n)\right] \overset{m(v)\le m'(v)}{\ge} \\ &\sum_{v\in\Upsilon_g}\left[W_v^{DPS}(s_g) - \min(W_v^{MPS}(s_g), \sum_{n=1}^{m'(v)} l_v^n)\right] \overset{(22)}{=} \\ &\sum_{v\in\Upsilon_g} \mathrm{lag}_v\langle d_i^j\rangle(s_g) \overset{(23)}{=} \sum_{v\in\Upsilon_g} \mathrm{lag}_v(s_g).\end{aligned} \tag{83}$$

Using all the inequalities proved so far, we get

$$\begin{aligned} s_g - d_g^l &\overset{(80)+(81)+(82)}{\le} \\ \frac{\Lambda(k-1)-\sum_{v\in\Upsilon_g}\mathrm{lag}_v\langle d_g^l\rangle(s_g)-\hat{l}_g}{M-\sum_{v\in\Upsilon_g}U_v} &\overset{(83)}{\le} \\ \frac{\Lambda(k-1)-\sum_{v\in\Upsilon_g}\mathrm{lag}_v(s_g)-\hat{l}_g}{M-\sum_{v\in\Upsilon_g}U_v}&. \end{aligned} \tag{84}$$

We can now compute our bound to $\mathrm{lag}_g(s_g)$ from (84). Considering that: 1) by Assumption 2, we replace J_g^l with a job of length L_g if $l_g^l < L_g$, 2) τ_g belongs to $\tau^{MPS}(\hat{J}, s_g)$ by item 2.a of Lemma 20, and 3) $s_g > r_g^l$ by item 2.b of Lemma 20, we have that $\hat{J}_g$ satisfies the conditions of Lemma 13 after setting $h = g$. As a consequence,

$$\mathrm{lag}_g(s_g) \overset{\text{(41) with } h=g}{\le} (s_g - d_g^l)U_g + \hat{l}_g \overset{(84)}{\le} U_g\frac{\Lambda(k-1)-\sum_{v\in\Upsilon_g}\mathrm{lag}_v(s_g)-\hat{l}_g}{M-\sum_{v\in\Upsilon_g}U_v} + \hat{l}_g. \tag{85}$$

□

8.5 Sub-step 2.2: Compute an upper bound to $\sum_{g\in\hat{\tau}}\mathrm{lag}_g(s_g)$ in open form

To prove the upper bound to $\sum_{g\in\hat{\tau}}\mathrm{lag}_g(s_g)$ stated in Lemma 25, we use the following (algebraic) properties of the coefficients M_g.

Lemma 27 *The following two relations holds:*

$$\forall g \in \{1,2,\ldots,G+1\} \quad M_g > 0, \tag{86}$$

$$\forall Q \in \{1,2,\ldots,G\}, \forall g \in \{1,2,\ldots,Q\} \quad 1 - M_{Q+1}\sum_{v=g}^{Q}\frac{U_v}{M_v M_{v+1}} = \frac{M_{Q+1}}{M_g}. \tag{87}$$

In particular, the last inequality implies, after setting $g = Q$,

$$1 - \frac{U_Q}{M_Q} = \frac{M_{Q+1}}{M_Q}. \tag{88}$$

Proof As for (86), we can write

$$\begin{aligned} M_g \overset{(74)}{=} M - \sum_{v=1}^{g-1} U_v \overset{g\leq G+1}{\geq} M - \sum_{v=1}^{G+1-1} U_v = M - \sum_{v=1}^{G} U_v \overset{U_v\leq 1}{\geq} M - \sum_{v=1}^{G} 1 = \\ M - G \overset{(58)}{\geq} M - (\lceil U_{sum} \rceil - 1) \geq M - (M-1) = 1 > 0. \end{aligned} \tag{89}$$

Instead, to prove (87), we proceed by induction on g, in *reverse* order, i.e., as a base case we consider $g = Q$, and, as an inductive step, we prove that, if the thesis holds for a given $g \in \{2,3,4,\ldots,Q\}$, then it holds for $g-1$ as well.

Base case. If $g = Q$, then the LHS of (87) becomes:

$$1 - M_{Q+1}\frac{U_Q}{M_Q M_{Q+1}} = 1 - \frac{U_Q}{M_Q} = \frac{M_Q - U_Q}{M_Q} \overset{(75)}{=} \frac{M_{Q+1}}{M_Q}. \tag{90}$$

Inductive step. We want to prove that

$$\begin{aligned} &\forall g \in \{2,,\ldots,Q\} \\ &1 - M_{Q+1}\sum_{v=g}^{Q}\frac{U_v}{M_v M_{v+1}} = \frac{M_{Q+1}}{M_g} \implies 1 - M_{Q+1}\sum_{v=g-1}^{Q}\frac{U_v}{M_v M_{v+1}} = \frac{M_{Q+1}}{M_{g-1}} \end{aligned} \tag{91}$$

Manipulating the LHS of the consequent of the above implication, we have

$$\begin{aligned} 1 - M_{Q+1}\sum_{v=g-1}^{Q}\frac{U_v}{M_v M_{v+1}} = 1 - M_{Q+1}\sum_{v=g}^{Q}\frac{U_v}{M_v M_{v+1}} - \frac{M_{Q+1}U_{g-1}}{M_{g-1}M_g} \overset{\text{Inductive hypothesis}}{=} \\ \frac{M_{Q+1}}{M_g} - \frac{M_{Q+1}U_{g-1}}{M_{g-1}M_g} = M_{Q+1}\frac{M_{g-1}-U_{g-1}}{M_{g-1}M_g} \overset{(75)}{=} M_{Q+1}\frac{M_g}{M_{g-1}M_g} = \frac{M_{Q+1}}{M_{g-1}}. \end{aligned} \tag{92}$$

□

We can now prove Lemma 25, which states that

$$\sum_{g=1}^{G}\text{lag}_g(s_g) \leq M_{G+1}\cdot\left(\Lambda(k-1)\sum_{g=1}^{G}\frac{U_g}{M_g M_{g+1}} - \sum_{g=1}^{G}\frac{U_g\Delta_g}{M_g M_{g+1}} + \sum_{g=1}^{G}\frac{L_g}{M_g}\right). \tag{93}$$

Proof (Lemma 25) The idea to prove the bound is to upper-bound each addend of the sum in the LHS of (93) using (73), starting from the last addend downward. Actually, to prove the thesis, we prove, by induction, a stronger property, i.e., that, for all integers $q \in \{0,1,\ldots,G\}$,

$$\begin{aligned} \sum_{g=1}^{G}\text{lag}_g(s_g) \leq \frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q}\text{lag}_g(s_g) + \Lambda(k-1)\sum_{g=q+1}^{G}\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}} + \\ - \sum_{g=q+1}^{G}\Delta_g\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}} + \sum_{g=q+1}^{G}L_g\frac{M_{G+1}}{M_g}. \end{aligned} \tag{94}$$

Before proving (94), we prove that (93) does follow from (94). To this purpose, if we set $q = 0$ in (94), and recall that we assume $\sum_{i=n_1}^{n_2} x = 0$ if $n_1 > n_2$, we get

$$\sum_{g=1}^{G} \mathrm{lag}_g(s_g) \leq$$
$$\frac{M_{G+1}}{M_{q+1}} \cdot 0 + \Lambda(k-1) \sum_{g=1}^{G} \frac{U_g}{M_g} \frac{M_{G+1}}{M_{g+1}} - \sum_{g=1}^{G} \Delta_g \frac{U_g}{M_g} \frac{M_{G+1}}{M_{g+1}} + \sum_{g=1}^{G} L_g \frac{M_{G+1}}{M_g} = \tag{95}$$
$$M_{G+1} \cdot \left(\Lambda(k-1) \sum_{g=1}^{G} \frac{U_g}{M_g M_{g+1}} - \sum_{g=1}^{G} \frac{U_g \Delta_g}{M_g M_{g+1}} + \sum_{g=1}^{G} \frac{L_g}{M_g} \right).$$

We prove now (94) by induction. We proceed in *reverse* order, that is, as a base case we prove that (94) holds for $q = G$ (the maximum possible value for q), while as an inductive step we prove that if (94) holds for an integer $q \in \{1, \ldots, G\}$, then it holds for $q - 1$.

Base case: $q = G$. We have that, if $q = G$, then (94) becomes the trivial inequality

$$\sum_{g=1}^{G} \mathrm{lag}_g(s_g) \leq \sum_{g=1}^{G} \mathrm{lag}_g(s_g). \tag{96}$$

Inductive step: we want to prove that, if (94) holds for an integer $q \in \{1, \ldots, G\}$ (inductive hypothesis), then it holds for $q - 1$ as well. To this purpose, first we compute the following upper bound to the first term in (94), where the inequality

$M_{q+1}/M_q > 0$ used in one of the steps follows from (86):

$$
\begin{aligned}
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q}\mathrm{lag}_g(s_g) &= \frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\frac{M_{G+1}}{M_{q+1}}\mathrm{lag}_q(s_q) && \overset{(73)}{\leq}\\
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\frac{M_{G+1}}{M_{q+1}}\left(U_q\frac{\Lambda(k-1)-\sum_{g=1}^{q-1}\mathrm{lag}_g(s_q)-\hat{l}_q}{M_q}+\hat{l}_q\right) && \overset{(3)}{=}\\
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\frac{M_{G+1}}{M_{q+1}}\left(U_q\frac{\Lambda(k-1)-\sum_{g=1}^{q-1}(\mathrm{lag}_g(s_g)+\mathrm{lag}(s_g,s_q))-\hat{l}_q}{M_q}+\hat{l}_q\right) && \overset{(76)}{=}\\
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\frac{M_{G+1}}{M_{q+1}}\left(U_q\frac{\Lambda(k-1)-\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)-\Delta_q-\hat{l}_q}{M_q}+\hat{l}_q\right) && \overset{\text{rearranging}}{=}\\
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\frac{M_{G+1}}{M_{q+1}}\left(U_q\frac{\Lambda(k-1)-\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)-\Delta_q}{M_q}+\hat{l}_q(1-\tfrac{U_q}{M_q})\right) && \overset{\substack{(88)\text{ with}\\ Q=G=q}}{=}\\
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\frac{M_{G+1}}{M_{q+1}}\left(U_q\frac{\Lambda(k-1)-\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)-\Delta_q}{M_q}+\hat{l}_q\frac{M_{q+1}}{M_q}\right) && \overset{\substack{\hat{l}_q\leq L_q\text{ and}\\ M_{q+1}/M_q>0}}{\leq}\\
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\frac{M_{G+1}}{M_{q+1}}\left(U_q\frac{\Lambda(k-1)-\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)-\Delta_q}{M_q}+L_q\frac{M_{q+1}}{M_q}\right) && \overset{\text{rearranging}}{=}\\
\frac{M_{G+1}}{M_{q+1}}\left(1-\frac{U_q}{M_q}\right)\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\Lambda(k-1)\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}+ &&\\
-\Delta_q\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}+L_q\frac{M_{G+1}}{M_{q+1}}\frac{M_{q+1}}{M_q} && \overset{\substack{(88)\text{ with}\\ Q=G=q}}{=}\\
\frac{M_{G+1}}{M_{q+1}}\frac{M_{q+1}}{M_q}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\Lambda(k-1)\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}+ &&\\
-\Delta_q\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}+L_q\frac{M_{G+1}}{M_{q+1}}\frac{M_{q+1}}{M_q} && \overset{\text{simplifying}}{=}\\
\frac{M_{G+1}}{M_q}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\Lambda(k-1)\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}-\Delta_q\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}+L_q\frac{M_{G+1}}{M_q}. &&
\end{aligned}
\tag{97}
$$

We can now write

$$
\begin{aligned}
\sum_{g=1}^{G}\mathrm{lag}_g(s_g) &\overset{\substack{\text{Inductive}\\ \text{hypothesis}}}{\leq}\\
\frac{M_{G+1}}{M_{q+1}}\sum_{g=1}^{q}\mathrm{lag}_g(s_g)+\Lambda(k-1)\sum_{g=q+1}^{G}\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}}+ &\\
-\sum_{g=q+1}^{G}\Delta_g\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}}+\sum_{g=q+1}^{G}L_g\frac{M_{G+1}}{M_g} &\overset{(97)}{\leq}\\
\frac{M_{G+1}}{M_q}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+ &\\
+\Lambda(k-1)\sum_{g=q+1}^{G}\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}}+\Lambda(k-1)\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}+ &\\
-\sum_{g=q+1}^{G}\Delta_g\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}}-\Delta_q\frac{U_q}{M_q}\frac{M_{G+1}}{M_{q+1}}+ &\\
+\sum_{g=q+1}^{G}L_g\frac{M_{G+1}}{M_g}+L_q\frac{M_{G+1}}{M_q} &\overset{\text{rearranging}}{=}\\
\frac{M_{G+1}}{M_q}\sum_{g=1}^{q-1}\mathrm{lag}_g(s_g)+\Lambda(k-1)\sum_{g=q}^{G}\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}}+ &\\
-\sum_{g=q}^{G}\Delta_g\frac{U_g}{M_g}\frac{M_{G+1}}{M_{g+1}}+\sum_{g=q}^{G}L_g\frac{M_{G+1}}{M_g}. &
\end{aligned}
\tag{98}
$$

□

8.6 Sub-step 2.3: Compute an upper bound to the total lag in open form

In this sub-step we prove Lemma 26, i.e., the following upper bound:

$$\sum_{h\in\tau(\hat{J},b_k)} \text{lag}_h(b_k) \leq M_{G+1}\cdot\left(\Lambda(k-1)\sum_{g=1}^{G}\frac{U_g}{M_gM_{g+1}}+\sum_{g=1}^{G}\frac{L_g}{M_g}\right). \tag{99}$$

As we already said, we prove this bound by adding $\sum_{g\in\hat{\tau}}\text{lag}_g(s_g,b_k)$ to the bound (77). As can be seen, the only difference between the RHS of (99) and the RHS of (77) is the absence of the term $-M_{G+1}\sum_{g=1}^{G}U_g\frac{\Delta_g}{M_gM_{g+1}}$ in the RHS of (99). To prove that this term cancels from the RHS of (77) after adding $\sum_{g\in\hat{\tau}}\text{lag}_g(s_g,b_k)$, we use the following property.

Lemma 28 *The following inequality holds:*

$$\forall Q\in\{0,1,\ldots,G\}\quad \sum_{g=1}^{Q-1} lag_g(s_g,s_Q)-M_{Q+1}\sum_{g=1}^{Q}U_g\frac{\sum_{v=1}^{g-1}lag_v(s_v,s_g)}{M_gM_{g+1}}\leq 0. \tag{100}$$

The above inequality holds for purely algebraic reasons, and its proof is relatively long. We report this proof in the appendix to not break the main flow. We can now prove Lemma 26.

Proof (Lemma 26) We prove the thesis in two steps. First, we substitute, in (72), the sum $\sum_{g\in\hat{\tau}}\text{lag}_g(s_g)$ with the RHS of (77), and the sum $\sum_{g\in\hat{\tau}}\text{lag}_g(s_g,b_k)$ with the following upper bound. Second, we prove that the sum of the RHS of (77) and the following upper bound is smaller than or equal to the RHS of (99). As for the upper bound to $\sum_{g\in\hat{\tau}}\text{lag}_g(s_g,b_k)$, we can write the following relations:

$$\begin{aligned}\sum_{g\in\hat{\tau}}\text{lag}_g(s_g,b_k)&\overset{(2)}{=}\sum_{g=1}^{G}\text{lag}_g(s_g,s_G)+\sum_{g=1}^{G}\text{lag}_g(s_G,b_k)\\ \sum_{g=1}^{G}\text{lag}_g(s_g,s_G)&=\sum_{g=1}^{G-1}\text{lag}_g(s_g,s_G)+\text{lag}_G(s_G,s_G)\\ &\sum_{g=1}^{G-1}\text{lag}_g(s_g,s_G).\end{aligned}\quad\begin{aligned}&\overset{\substack{(59)\text{ with }g=G+1,v=g,\\ t_v=s_G,\bar{t}=b_k}}{\leq}\\ &\overset{(1)}{=}\end{aligned} \tag{101}$$

Using (101), we can write

$$\begin{aligned}\sum_{h\in\tau(\hat{J},b_k)}\text{lag}_h(b_k)&\overset{(72)}{\leq}\sum_{g\in\hat{\tau}}\text{lag}_g(s_g)+\sum_{g\in\hat{\tau}}\text{lag}_g(s_g,b_k)&&\overset{(101)}{\leq}\\ &\sum_{g\in\hat{\tau}}\text{lag}_g(s_g)+\sum_{g=1}^{G-1}\text{lag}_g(s_g,s_G)&&\overset{(77)+(76)}{\leq}\\ M_{G+1}&\left(\Lambda(k-1)\sum_{g=1}^{G}\frac{U_g}{M_gM_{g+1}}-\sum_{g=1}^{G}U_g\frac{\sum_{v=1}^{g-1}\text{lag}_v(s_v,s_g)}{M_gM_{g+1}}+\sum_{g=1}^{G}\frac{L_g}{M_g}\right)+\\ &+\sum_{g=1}^{G-1}\text{lag}_g(s_g,s_G)&&\overset{\text{rearranging}}{=}\\ &M_{G+1}\left(\Lambda(k-1)\sum_{g=1}^{G}\frac{U_g}{M_gM_{g+1}}+\sum_{g=1}^{G}\frac{L_g}{M_g}\right)+\\ &+\sum_{g=1}^{G-1}\text{lag}_g(s_g,s_G)-M_{G+1}\sum_{g=1}^{G}U_g\frac{\sum_{v=1}^{g-1}\text{lag}_v(s_v,s_g)}{M_gM_{g+1}}&&\overset{\substack{(100)\\ \text{with }Q=G}}{\leq}\\ &M_{G+1}\left(\Lambda(k-1)\sum_{g=1}^{G}\frac{U_g}{M_gM_{g+1}}+\sum_{g=1}^{G}\frac{L_g}{M_g}\right).\end{aligned} \tag{102}$$

□

8.7 Sub-step 2.4: Compute an upper bound to the total lag and to the tardiness in closed form

In this last sub-step we prove, firstly, Lemma 5, i.e., that $\Lambda(k) \leq B$ holds for all k, with B equal to the RHS of (33). To achieve this goal, we start by proving (71), i.e., the implication that lets the inductive step ($\Lambda(k-1) \leq B \Rightarrow \Lambda(k) \leq B$) hold. In particular, as we already said, we prove (71) by focusing on a generic job portion $\hat{J} \subseteq J_i^j$ for which the k-th non-growing-Lag interval is the last non-growing-Lag interval. In this respect, the problem with computing a quantity for which (71) holds (after replacing $\hat{J}_k$ with $\hat{J}$) is that, according to (78) and depending on the value of $\Lambda(k-1)$, the sum $\sum_{h \in \tau(\hat{J}, b_k)} \text{lag}_h(b_k)$ may be *higher* than $\Lambda(k-1)$. Fortunately, as we prove in the next lemma, and still according to (78), there exists a *saturation* value, which we denote as Λ, such that

$$\Lambda(k-1) \geq \Lambda \implies \sum_{h \in \tau(\hat{J}, b_k)} \text{lag}_h(b_k) \leq \Lambda(k-1). \tag{103}$$

Therefore, intuitively, if $\Lambda(k)$ reaches or overcomes the threshold Λ, then it cannot grow any more with k, until it becomes again lower than or equal to Λ. This is, informally, the property we use to prove (71). In the next lemma we prove that such a threshold Λ is equal to the quantity already denoted by Λ in Lemma 5.

Lemma 29 *If Λ is defined as in* (32)*, then* (103) *holds for all $k > 1$ and for every portion $\hat{J}$ for which $[b_k, f_k)$ is the last non-growing-Lag interval.*

Proof We prove the thesis through the following derivations:

$$\begin{aligned}
\sum_{h \in \tau(\hat{J}, b_k)} \text{lag}_h(b_k) &\overset{(78)}{\leq} M_{G+1} \cdot \left(\Lambda(k-1) \sum_{g=1}^{G} \frac{U_g}{M_g M_{g+1}} + \sum_{g=1}^{G} \frac{L_g}{M_g} \right) \leq \Lambda(k-1) \iff \\
\Lambda(k-1) \cdot \left(M_{G+1} \sum_{g=1}^{G} \frac{U_g}{M_g M_{g+1}} - 1 \right) + M_{G+1} \sum_{g=1}^{G} \frac{L_g}{M_g} &\leq 0 \overset{\substack{(87) \text{ with} \\ Q=G, \\ g=1, \\ \nu=g}}{\iff} \\
-\Lambda(k-1) \frac{M_{G+1}}{M_1} + M_{G+1} \sum_{g=1}^{G} \frac{L_g}{M_g} &\leq 0 \overset{(74)}{\iff} \\
-\Lambda(k-1) \frac{M_{G+1}}{M} + M_{G+1} \sum_{g=1}^{G} \frac{L_g}{M_g} \leq 0 &\overset{(86)}{\iff} -\Lambda(k-1) \frac{1}{M} + \sum_{g=1}^{G} \frac{L_g}{M_g} \leq 0 \iff \\
-\frac{\Lambda(k-1)}{M} \leq -\sum_{g=1}^{G} \frac{L_g}{M_g} &\iff \frac{\Lambda(k-1)}{M} \geq \sum_{g=1}^{G} \frac{L_g}{M_g} \iff \Lambda(k-1) \geq M \sum_{g=1}^{G} \frac{L_g}{M_g}.
\end{aligned} \tag{104}$$

As for the RHS of the last inequality, we can consider the following true proposition:

$$G \overset{(58)}{\leq} \lceil U_{sum} \rceil - 1 \quad \wedge \quad \forall g \leq G, \frac{L_g}{M_g} \overset{\substack{L_g > 0, \\ (86)}}{>} 0. \tag{105}$$

We can then continue our derivations as follows:

$$\begin{aligned} \Lambda(k-1) \geq M \sum_{g=1}^{G} \frac{L_g}{M_g} \overset{(105)}{\Longleftarrow} \Lambda(k-1) \geq M \sum_{g=1}^{\lceil U_{sum} \rceil - 1} \frac{L_g}{M_g} \overset{\text{Def. 1, (4)}}{\Longleftarrow} \\ \Lambda(k-1) \geq M \cdot \max_{\substack{\tau' \subseteq \tau \text{ and} \\ |\tau'| = \lceil U_{sum} \rceil - 1}} \sum_{g=1}^{|\tau'|} \frac{L_{g'}}{M_g^{\tau'}} \overset{(32)}{=} \Lambda. \end{aligned} \tag{106}$$

□

Using the previous lemma, we can finally prove Lemma 5.

Proof (Lemma 5) We prove the thesis, i.e., that $\Lambda(k) \leq B$ for all k, by induction. Since B differs from the RHS of (7) only in that $C_{\nu'}$ is replaced by $L_{\nu'}$ and Γ is replaced by Λ, we have that $B \geq 0$ follows from exactly the same steps as the proof of Lemma 2, after substituting $L_{\nu'}$ for $C_{\nu'}$ and Λ for Γ.

Base case: $k = 1$. According to (25), $\Lambda(1) = \sum_{h \in \tau(\hat{J}_1, b_1)} \text{lag}_h(b_1)$. Then, by item 1 in Lemma 20 with $\hat{J} = \hat{J}_1$, we have $\Lambda(1) = \sum_{h \in \tau(\hat{J}_1, b_1)} \text{lag}_h(b_1) \leq 0$. Therefore $\Lambda(1) \leq B$ holds because $B \geq 0$.

Inductive step: we prove that, if $\Lambda(k-1) \leq B$ holds, then $\Lambda(k) \leq B$ holds. We consider two alternatives. First, $B < \Lambda$, with Λ defined as in Lemma 29. In this case, considering also the inductive hypothesis, we have $\Lambda(k-1) \leq B < \Lambda$, and hence $\Lambda(k-1) < \Lambda$. Then we have the following sequence of relations, where the last inequality follows from the fact that, according to the second inequality in (58), i.e., $|\hat{\tau}| \leq \lceil U_{sum} \rceil - 1$, plus Definition 1 and (4), the expression in the LHS of the last line yields the maximum possible value for the expression in the penultimate line:

$$\begin{aligned} \sum_{h \in \tau(\hat{J}, b_k)} \text{lag}_h(b_k) \overset{(78)}{\leq} M_{G+1} \cdot \left(\Lambda(k-1) \sum_{g=1}^{G} \frac{U_g}{M_g M_{g+1}} + \sum_{g=1}^{G} \frac{L_g}{M_g} \right) \overset{\substack{\Lambda(k-1) < \Lambda, \\ U_g \geq 0, \ (86)}}{<} \\ M_{G+1} \cdot \left(\Lambda \sum_{g=1}^{G} \frac{U_g}{M_g M_{g+1}} + \sum_{g=1}^{G} \frac{L_g}{M_g} \right) \overset{(58)}{=} \\ M_{|\hat{\tau}|+1} \cdot \left(\Lambda \sum_{g=1}^{|\hat{\tau}|} \frac{U_g}{M_g M_{g+1}} + \sum_{g=1}^{|\hat{\tau}|} \frac{L_g}{M_g} \right) \overset{\text{(58) + Def. 1 + (4)}}{\leq} \\ \max_{\substack{\tau' \subseteq \tau \text{ and} \\ |\tau'| \leq \lceil U_{sum} \rceil - 1}} \left[M_{|\tau'|+1}^{\tau'} \cdot \left(\Lambda \sum_{g=1}^{|\tau'|} \frac{U_{g'}}{M_g^{\tau'} M_{g+1}^{\tau'}} + \sum_{g=1}^{|\tau'|} \frac{L_{g'}}{M_g^{\tau'}} \right) \right] \quad = \quad B. \end{aligned} \tag{107}$$

The thesis, i.e., $\Lambda(k) \leq B$, follows from replacing the bounds $\Lambda(k-1) \leq B$ (which holds by the inductive hypothesis) and (107) in the first and in the second argument of the last max function in (69); that is, restarting from that last max function,

$$\Lambda(k) \overset{(69)}{=} \max \left[\Lambda(k-1), \sum_{h \in \tau(\hat{J}_k, b_k)} \text{lag}_h\langle \hat{d}_k \rangle(b_k) \right] \overset{\substack{\text{Inductive Hp.} \\ + \\ (107)}}{\leq} \max[B, B] = B. \tag{108}$$

The second case is that $B \geq \Lambda$. For this case we consider two further alternatives: $\Lambda(k-1) < \Lambda$ and $\Lambda(k-1) \geq \Lambda$. For the first alternative, the thesis follows from exactly the same steps as for the first case above. For the second alternative, i.e., if $\Lambda(k-1) \geq \Lambda$ holds, we have

$$\sum_{h \in \tau(\hat{f}, b_k)} \text{lag}_h(b_k) \overset{\substack{\text{Lemma 29,} \\ \text{(103) and} \\ \Lambda(k-1) \geq \Lambda}}{\leq} \Lambda(k-1) \overset{\substack{\text{Inductive} \\ \text{hypothesis}}}{\leq} B. \tag{109}$$

Similarly to the first case, the thesis, i.e. $\Lambda(k) \leq B$, follows from replacing $\Lambda(k-1) \leq B$ (which holds by the inductive hypothesis) and (109) in (69). □

We can now easily prove the harmonic bound.

Proof (Theorem 1) Substituting (33) in (31), we get

$$f_i^j - d_i^j \leq \frac{1}{M} \cdot \max_{\substack{\tau' \subseteq \tau \text{ and} \\ |\tau'| \leq \lceil U_{sum} \rceil - 1}} \left[M^{\tau'}_{|\tau'|+1} \cdot \left(\Lambda \sum_{g=1}^{|\tau'|} \frac{U_{g'}}{M^{\tau'}_g M^{\tau'}_{g+1}} + \sum_{g=1}^{|\tau'|} \frac{L_{g'}}{M^{\tau'}_g} \right) \right] + \\ + \frac{M-1}{M} C_i, \tag{110}$$

with the assumption that the denominator in the first fraction, namely $\frac{1}{M}$, is a speed. Moving from this assumption to the assumption that this denominator is a pure number, we have to replace $L_{g'}$ with $C_{g'}$, and Λ with Γ in (110). This yields (7) according to (6).

To complete the proof, we note that we obtained (110) from (33) and (31), and we proved (33) and (31) using Assumption 2. Nevertheless, having derived (7) from (110) is enough for the bound (7) to hold also if Assumption 2 does not hold. In fact, denoting by $\hat{s}$ the start time of the last portion of J_i^j, by Lemma 23 we have that the maximum possible value for $\hat{s}$, in the case Assumption 2 does not hold, is not higher than the maximum possible value for $\hat{s}$ in the case Assumption 2 holds. The same property therefore holds also for the maximum possible value of f_i^j. Thus, if the difference $f_i^j - d_i^j$ is at most equal to the RHS of (7) under Assumption 2, then this difference is at most equal to the same quantity also if Assumption 2 does not hold.

□

9 Experiments

In this section we compare the tightness of the harmonic bound with that of existing bounds for implicit-deadline tasks. To obtain the results reported in this section, we simulated the execution of random task sets, generated according to the distributions of utilizations and periods considered in previous work about tardiness or lateness (e.g., Erickson and Anderson (2012); Ward et al (2013); Erickson et al (2014)). In

the next paragraphs we describe how we generated the task sets, simulated their execution and measured tightness. Then we report our results. Both the code used in the experiments and our full results can be found in (Experiment-scripts (2014)[9]).

Systems and task sets. We generated task sets for systems with two to eight processors, as eight has been inferred as the largest number of processors for which G-EDF is an effective solution to provide SRT guarantees (Bastoni et al (2010)). In particular, for each number of processors M, we considered task sets with total utilizations U_{sum} in the range $[M/2, M]$, increasing in steps of $M/10$. Regarding individual task utilizations, we considered three uniform and three bimodal distributions. As for uniform distributions, we considered a *light*, a *medium* and a *heavy* one, with task utilizations distributed in, respectively, in $[0.001, 0.1]$, $[0.01, 0.99]$ and $[0.5, 0.99]$. Instead, in the three bimodal distributions, task utilizations were uniformly distributed in either $[0.01, 0.5]$ or $[0.5, 0.99]$, with probabilities, respectively, $8/9$ and $1/9$, $6/9$ and $3/9$, and $4/9$ and $5/9$. We call, respectively, *light*, *medium* and *heavy* these three bimodal distributions. Finally, we considered three possible uniform distributions for task periods, denoted as *short*, *moderate* and *long*, and in the ranges $[3ms, 33ms]$, $[10ms, 100ms]$ and $[50ms, 250ms]$.

We generated 1000 sets of implicit-deadline periodic tasks for every: number of processors M in $[2, 8]$, total utilization U_{sum} in $[M/2, M]$, combination of distributions of task utilizations and periods. For brevity, hereafter we denote as just *group* of task sets, each group of 1000 task sets generated with the same combination of these parameters.

Simulation. We simulated the execution of the task sets using RTSIM (2011). We let each simulation last for the maximum duration supported by RTSIM, which happened to be, for any task set, at least 8K times as long as the longest task period.

Tardiness Bounds. We considered the harmonic bound (HARM) and the three tardiness bounds for G-EDF computed, respectively, with the analysis of Devi and Anderson (2008) (DA), the CVA analysis using the PP Reduction Rule (CVA), and the CVA analysis using the alternative optimization rule proved by Devi and Anderson (2008), as defined by Erickson et al (2010) (CVA2). We calculated the values of the latter three bounds using SchedCAT (2014).

Tightness index and normalized error. Given: a tardiness bound, a simulation run for a task set, and a task in the set, we define as *observed tightness index* of the bound for that task, the ratio between the value of the bound for that task and the maximum tardiness that that task experiences in the run. We use this index as a tightness measure in our results, because this index is unbiased with respect to a change of the time scale, i.e., the value of the index does not change if the execution time, the period and the arrival times of the jobs of every task are all multiplied by a common factor. This invariant does not hold, for example, for the difference between the value of a bound and the maximum observed tardiness.

A problem of the observed tightness index is however that it is well-defined only for tasks that experience a non-null tardiness. Fortunately, there happened to be a clear distinction among groups of task sets in terms of experienced tardiness: for

[9] All experiments can be repeated by applying two patches to SchedCAT (2014), replacing one file in RTSIM (2011), and running an *ad-hoc* script.

each group of task sets, either all the tasks of all the task sets happened to experience a non-null tardiness, or almost all the tasks of all the task sets experienced a null tardiness. In particular, as we highlight when discussing results, in the second case all bounds happened to be remarkably loose.

To measure the tightness of the bounds also in the second case, we introduce a second measure of tightness too. This second metric may be less robust with respect to scaling issues than the observed tightness index, but is well-defined also for tasks that experience a null tardiness. We call this metric *observed normalized error*, and we define it as the difference between the value of the bound and the actual maximum tardiness experienced by the task, divided by the period of the task. The idea behind this normalization is that the period of a task, especially with implicit deadlines, represents somehow a reference time interval for understanding how tolerable a given error on the tardiness is for that task. In other words, a given absolute error on the tardiness of a task is likely to be less relevant for a task with a large period than for a task with a short period. The purpose of the normalization is of course also to try to offset the above-discussed time-scale problems.

Measures and statistics. At the end of each simulation run we computed, for each bound and for each task, the observed tightness index and the normalized error of the bound for that task. Then, for each bound and each group of task sets, we computed the minimum and the average values of the observed tightness indexes and of the normalized errors for that bound, over all the tasks of all the task sets in the group. For brevity, hereafter we call these four quantities the *minimum* and the *average tightness index*, and the *minimum* and the *average normalized error* for that group of task sets. We also computed the 95% confidence interval for the average tightness index and the average normalized error. For both average values, and over all the groups of task sets, the confidence interval was never above 15% of the average, and, for most groups of task sets, it was below 5%. For this reason, to reduce clutter we do not show also confidence intervals in the next figures.

For both the minimum and the average tightness indexes, it is worth noting that lower values than the ideal value, one, are of course not allowed. In this respect, according to the definition of tightness reported in the introduction, the closer the minimum tightness index of a bound B is to one for at least one of the groups of task sets generated for a given number of processors, the closer the bound is to being tight for that number of processors. In view of these facts, given two bounds $B1$ and $B2$ with minimum tightness indexes I_{B1} and I_{B2} for a group of task sets G, we measure how tighter $B1$ is than $B2$ as a function of how closer I_{B1} is to one, and not to zero, with respect to I_{B2}. In formulas, and assuming that $I_{B2} > 1$ holds (otherwise it is enough to swap $B1$ and $B2$ and reverse the statement), we say that $B1$ is $x\%$ tighter or $x\%$ looser than $B2$ for G, if $x = 100 * |I_{B2} - I_{B1}|/(I_{B2} - 1)$, and $I_{B2} < I_{B1}$ (tighter) or $I_{B1} < I_{B2}$ (looser).

9.1 Results

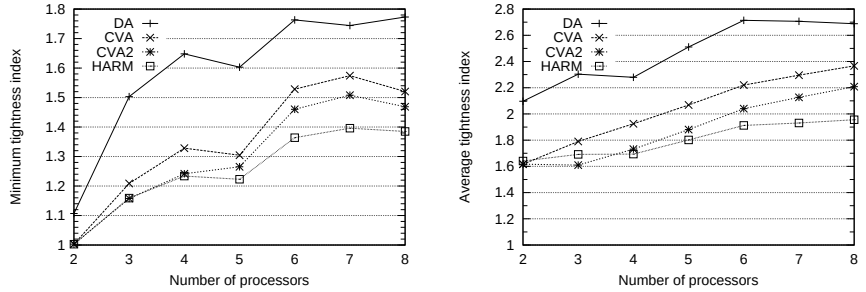
Full total utilization and non-light per-task utilizations. First we focus on task sets with a total utilization equal to M, and on all distributions of utilizations except for

light uniform distributions. Figures 6, 7, 8 and 9 show the minimum and the average tightness indexes of the four bounds for four representative cases in this first subset of groups of tasks. Specifically, these figures show the performance of the bounds for four different combinations of medium or heavy utilizations, and of short or long periods. The figures highlight the following general results, which hold not only for the combinations of parameters considered in the figures, but, actually, for all the groups of tasks in this subset (full results can be found in (Valente (2014)):

- The relative order among both the minimum and the average tightness indexes of the bounds is the same for every value of M, apart from $M = 3$, for which the average tightness index of HARM is slightly higher than that of CVA2. This happens also with some other groups of task sets (see (Valente (2014))).
- In accordance with the experimental results available in the literature (Ward et al (2013)), the minimum and average tightness indexes of DA are substantially larger than those of CVA and CVA2. In particular, DA is up to 40% looser than CVA2 (Figure 6).
- In all cases, both the minimum and the average tightness indexes of HARM are significantly lower than the corresponding tightness indexes of DA. In particular, HARM is up to 50% tighter than DA (Figure 6). This gap highlights the effectiveness of the lag-balance property, and is the reason behind the following positive results.
- Both the minimum and the average tightness indexes of HARM are always at least as low as those of the second best-performing bound, namely CVA2, apart from the above-mentioned cases, with $M = 3$, where the average tightness index of HARM is slightly higher than that of CVA2.
- HARM outperforms CVA2 from $M = 4$ on. In particular, the minimum and average tightness indexes of HARM become lower and lower than those of CVA2 as M increases, apart from some occasional fluctuation (i.e., the slope of HARM is smaller than the slope of CVA2).
- With $M = 8$, HARM is from 18% (Figure 6) to 29% (Figure 7) tighter than CVA2.

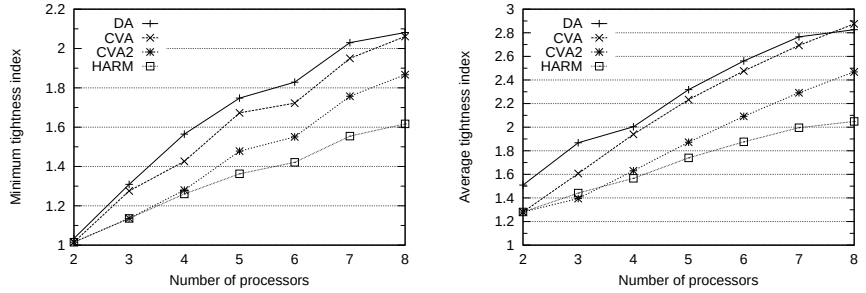
As a general consideration, both average and minimum tightness indexes do not always grow with M (Figure 6). In contrast, occasional fluctuations affect a few groups of task sets, as can be seen in the extended version of this paper (Valente (2014)). The reason is apparently just that *luckier* scenarios occur occasionally for some bounds (this issue may deserve further, non-trivial investigations).

Light per-task utilizations. Things change dramatically with uniform light utilizations. Fortunately, the tightness of tardiness bounds may not be very relevant in these cases, as we discuss after showing the performance of the bounds. Figure 10.a reports the minimum tightness index for a representative case for uniform light utilizations and total utilization equal to M (we comment on the average tightness index and Figure 10.b in a moment). First, with uniform light utilizations, the generated task sets quickly become very large as the number of processors increases. Because of this fact, the figure reports results only for $M \leq 6$, as with higher values of M it was unfeasible to compute the value of the harmonic bound for all the tasks. Regardless of this limitation, the figure clearly shows that all bounds become very loose if all task utilizations are light. Fluctuations become very large too.



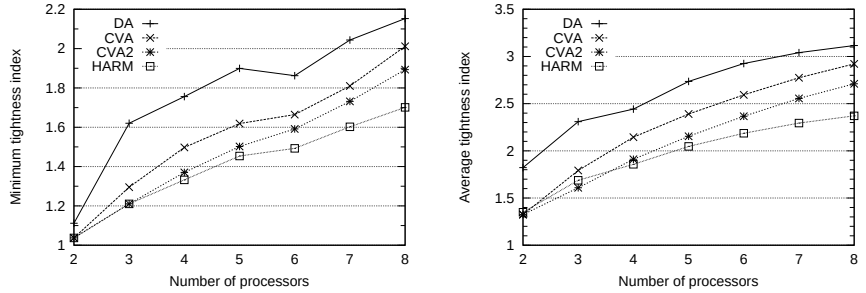
(a) Minimum tightness index (b) Average tightness index

Fig. 6: Minimum and average tightness index with bimodal medium utilizations, short periods and total utilization equal to M (lower is better, the ideal value is 1).



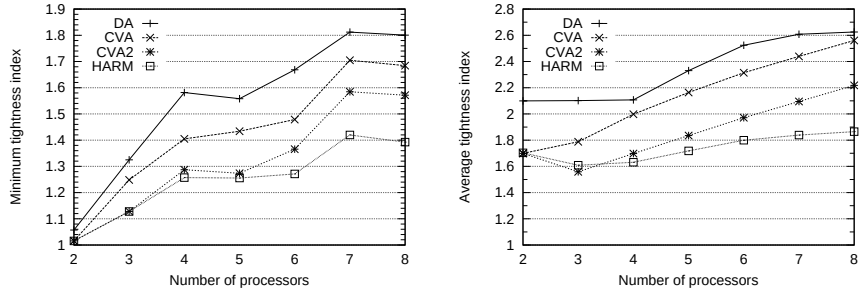
(a) Minimum tightness index (b) Average tightness index

Fig. 7: Minimum and average tightness index with uniform heavy utilizations, long periods and total utilization equal to M (lower is better, the ideal value is 1).



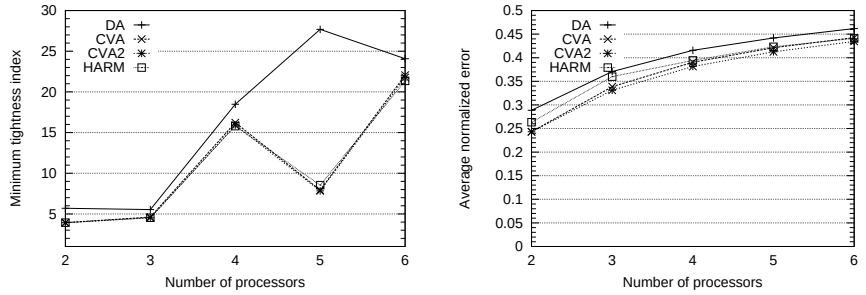
(a) Minimum tightness index (b) Average tightness index

Fig. 8: Minimum and average tightness index with bimodal medium utilizations, long periods and total utilization equal to M (lower is better, the ideal value is 1).



(a) Minimum tightness index (b) Average tightness index

Fig. 9: Minimum and average tightness index with uniform heavy utilizations, short periods and total utilization equal to M (lower is better, the ideal value is 1).



(a) Minimum tightness index (b) Average normalized error

Fig. 10: Minimum tightness index and average normalized error with uniform light utilizations, long periods and total utilization equal to M (lower is better, the ideal value is 1 for the tightness index and 0 for the normalized error). The maximum number of processors is limited to 6 for feasibility issues.

We cannot show the average tightness index because it is infinite for all bounds and values of M. In fact, in all runs, almost every task experiences a null tardiness, whereas, for every task, all bounds happen to be at least in the order of the overall maximum execution time for the task set. This result differs substantially with respect to the previous cases, where in all runs every task happens to experience a non-null tardiness. To show the average performance of the bounds also with uniform light utilizations, we resort to the average normalized error in Figure 10.b. The bounds are quite loose also on average: as M increases, the average error ranges from $1/4$ to almost $1/2$ of the period.

Fortunately, the case of light utilizations is exactly one of those for which tardiness bounds may not be very relevant, for the following two reasons. First, defining $U_{max} \equiv \max_{\tau_i \in \tau} U_i$, Goossens et al (2003) proved that G-EDF meets all deadlines for every implicit-deadline task set for which the total utilization U_{sum} satisfies the

following inequality:

$$U_{sum} \leq M - U_{max} \cdot (M-1). \tag{111}$$

This implies that, if U_{max} is very small, as it is the case with uniform light utilizations, then it is enough to keep the total utilization slightly below M to make sure that all deadlines are met with G-EDF.

Secondly, if partitioned EDF can be used instead of G-EDF (and if all utilizations are light), then all deadlines can be met at the price of an even lower loss of total utilization than with G-EDF. In more detail, the scenario in question is when there is no hindrance to partitioning tasks among processors, and scheduling each per-processor subset of tasks with EDF. In fact, given a generic task set with $U_{sum} \leq M$, consider the sum, say S, of the utilizations of the tasks that may fail to be accommodated in a feasible partitioning. If all task utilizations are very low, then the sum S can be equal at most to a very low fraction of the total utilization of the task set (as for the worst possible case, $S \leq U_{max} \cdot (M-1)$ holds).

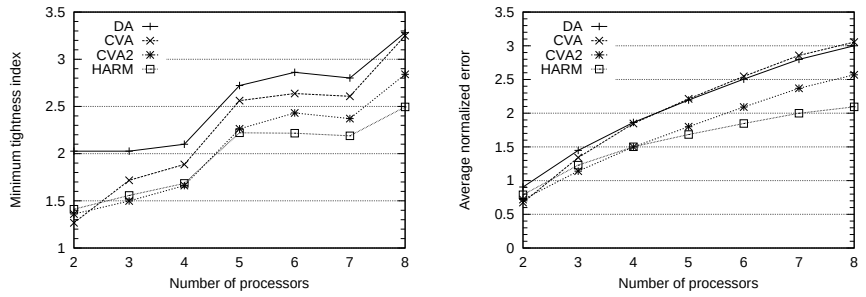
In addition, a partitioned scheme has a reduced overhead with respect to a global one. This increases the actual total utilization achievable. Since all tasks have a very low utilization, this increase may easily offset the above loss S. In the end, when all task utilizations are low, it may be possible to meet all deadlines with no or negligible loss in terms of total utilization.

Total utilization lower than M. Similar tightness problems occur, with all distributions of utilizations, if the total utilization U_{sum} is lower than M. Figure 11.a shows, e.g., the minimum tightness index for the same distributions of utilizations and periods as in Figure 7, but with $U_{sum} = 0.9 \cdot M$. As can be seen, after decreasing U_{sum} by just $0.1 \cdot M$, the minimum tightness index becomes much higher than with $U_{sum} = M$. In particular, all the bounds are quite loose for $M > 5$. As with uniform light utilizations, the average tightness index is infinite for all groups of task sets, bounds, and values of M. Then, also in this case, we show the average performance of the bounds through the average normalized error in Figure 11.b. Now the situation is much worse than with uniform light utilization, because the average normalized error ranges from about 0.7 to more than 3.

Although all bounds quickly become remarkably loose as M grows, it is worth noting that the harmonic bound outperforms the other bounds more and more, in terms of both minimum tightness index and average normalized error, as M increases beyond 5. In particular, the harmonic bound is the only one to preserve a minimum tightness index at most equal to 2.5, and an average normalized error at most equal to 2.1.

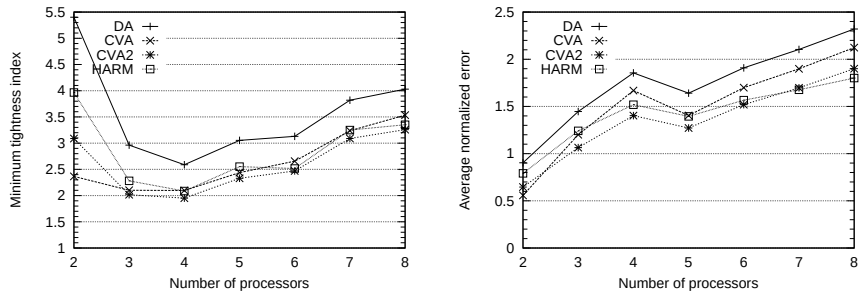
Going down to $U_{sum} = 0.8 \cdot M$, both the minimum tightness index and the average normalized error have again high values, as shown in Figure 12 (the average tightness index is of course again infinite). They also fluctuate more with M. Before commenting on the relative performance of the harmonic bound, we highlight that the situation becomes quite critical with $M = 2$, because the actual tardiness experienced by the tasks tends to be very small.

In this respect, if we further reduce the total utilization to $0.7 \cdot M$, then even the minimum tightness index becomes infinite for all bounds and groups of task sets with $M = 2$, except for CVA. As a consequence, to show the performance of the bounds



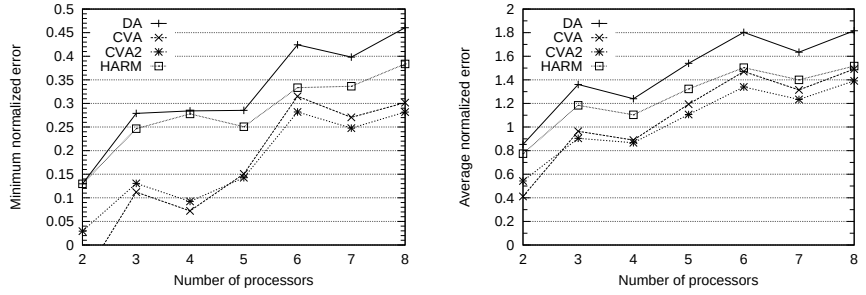
(a) Minimum tightness index (b) Average normalized error

Fig. 11: Minimum tightness index and average normalized error with uniform heavy utilizations, long periods and total utilization equal to $0.9 \cdot M$ (lower is better, the ideal value is 1 for the tightness index and 0 for the normalized error).



(a) Minimum tightness index (b) Average normalized error

Fig. 12: Minimum tightness index and average normalized error with uniform heavy utilizations, long periods and total utilization equal to $0.8 \cdot M$ (lower is better, the ideal value is 1 for the tightness index and 0 for the normalized error).



(a) Minimum normalized error (b) Average normalized error

Fig. 13: Minimum and average normalized error with uniform heavy utilizations, long periods and total utilization equal to $0.7 \cdot M$ (lower is better, the ideal value is 0).

for $U_{sum} = 0.7$, in Figure 13.a we report the minimum normalized error instead of the minimum tightness index. The index is negative for CVA with $M = 2$, because CVA is actually a lateness bound, and does take negative values for some task for $M = 2$.

Finally, Figure 13.b reports the average normalized error for $U_{sum} = 0.7$, and highlights that also in this case, as expected, the bounds are definitely loose. Regarding the trend of the average normalized errors, the latter decrease moving from Figure 11.b to Figure 13.b. The reason is simply that the values of all the bounds do decrease as U_{sum} decreases, while the tardiness of the tasks remains unchanged, i.e., equal to 0.

Although the actual problem is that all bounds are very loose, we can also point out that the performance of the harmonic bound degrades slightly more than that of the other bounds as U_{sum} decreases, and that the harmonic bound is no longer the best-performing one with $U_{sum} \leq 0.8$. This result complies with the fact that, as we already highlighted in the comments after the proof of Lemma 10 in Section 7.2, we computed the harmonic bound in a simplified way. On the flip side, this simplification lets the bound become explicitly looser as the ratio U_{sum}/M decreases.

With $U_{sum} = 0.6 \cdot M$, the performance of the bounds is about the same as for the case $U_{sum} = 0.7 \cdot M$. We do not show results also for $U_{sum} = 0.6 \cdot M$ and below, for similar reasons as for the above case of light utilizations. First, the probability that a generic task set meets (111) is not negligible with $U_{sum} = 0.6 \cdot M$ (unless the task set contains tasks with a very high utilization), and this probability increases as U_{sum} decreases. In addition, and probably even more relevant, there are partitioned scheduling algorithms, including variants of EDF itself, with which a task set with $U_{sum} = 0.6 \cdot M$ is very likely to be schedulable, while all task sets with $U_{sum} \leq 0.5 \cdot M$ are schedulable (Davis and Burns (2011)).

To sum up, tardiness bounds, and thus their tightness, may be little relevant for $U_{sum} \leq 0.6 \cdot M$. In contrast, there is a band of total utilizations of interest, ranging from about $0.7 \cdot M$ to about $0.9 \cdot M$, for which all bounds happen to be remarkably loose. Fortunately, the harmonic bound has room for improvement for this band of total utilizations, because, as we already pointed out above, it is currently computed in a simplified way that lets it become looser and looser and the ratio U_{sum}/M decreases.

10 Conclusion and future work

In this paper we showed how to compute a new tardiness bound for preemptive global EDF and implicit-deadline tasks, by integrating a lag-balance property, enjoyed by any work-conserving scheduling algorithm, with the approach used to compute one of the first tardiness bounds for G-EDF (Devi and Anderson (2008)). According to our experiments, the new bound, which we tagged as *harmonic*, is up to 50% tighter than the original bound obtained through the same approach (the maximum improvement is reached with $M = 8$ and a total utilization equal to M). As a consequence of this improvement, in spite of the fact that the original bound results to be, in the worst-case, 40% looser than the bounds proposed in the intervening years, the harmonic bound is up to 29% tighter than the best available bound.

Such a result may open new ways for obtaining tighter response-time or utilization bounds, with existing or new scheduling algorithms. As next steps, we plan to

generalize the harmonic bound to consider also non-preemptive global EDF and task sets with a lower total utilization than the system capacity. We also want to investigate more efficient algorithms for computing, or at least approximating the bound. In fact, the brute-force algorithm reported in this paper has an exponential running time, although it has proved to be feasible for all the task sets considered in the experiments, except for some of the cases where tardiness bounds are probably not very relevant.

Finally, in this paper we also highlighted a general negative result: with light distributions, as well as with total utilizations lower than the total system capacity, all bounds proved to be quite loose.

References

Anderson JH, Srinivasan A (2004) Mixed pfair/erfair scheduling of asynchronous periodic tasks. Journal of Computer and System Sciences 68(1):157 – 204, DOI 10.1016/j.jcss.2003.08.002, URL http://www.sciencedirect.com/science/article/pii/S0022000003001508

Baruah SK, Cohen NK, Plaxton CG, Varvel DA (1996) Proportionate progress: A notion of fairness in resource allocation. Algorithmica 15:600–625

Bastoni A, Brandenburg B, Anderson J (2010) An empirical comparison of global, partitioned, and clustered multiprocessor real-time schedulers. In: Proceedings of the 31st IEEE Real-Time Systems Symposium, pp 14–24

Brandenburg BB (2011) Scheduling and locking in multiprocessor real-time operating systems. PhD thesis, Chapel Hill, NC, USA, aAI3502550

Davis RI, Burns A (2011) A survey of hard real-time scheduling for multiprocessor systems. ACM Comput Surv 43(4):35:1–35:44, DOI 10.1145/1978802.1978814, URL http://doi.acm.org/10.1145/1978802.1978814

Devi UC, Anderson JH (2005) Tardiness bounds under global edf scheduling on a multiprocessor. In: RTSS, IEEE Computer Society, pp 330–341, URL http://dblp.uni-trier.de/db/conf/rtss/rtss2005.html#DeviA05

Devi UC, Anderson JH (2008) Tardiness bounds under global edf scheduling on a multiprocessor. Real-Time Systems 38(2):133–189

Erickson JP, Anderson JH (2012) Fair lateness scheduling: Reducing maximum lateness in g-edf-like scheduling. In: ECRTS, pp 3–12

Erickson JP, Devi U, Baruah SK (2010) Improved tardiness bounds for global edf. In: ECRTS, pp 14–23

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

Experiment-scripts (2014) Code used for experiments. URL http://algogroup.unimore.it/people/paolo/harmonic-bound/

Goossens J, Funk S, Baruah S (2003) Priority-driven scheduling of periodic task systems on multiprocessors. Real-time systems 25(2-3):187–205

Kenna CJ, Herman JL, Brandenburg BB, Mills AF, Anderson JH (2011) Soft real-time on multiprocessors: Are analysis-based schedulers really worth it? In: RTSS, IEEE Computer Society, pp 93–103, URL http://dblp.uni-trier.de/db/conf/rtss/rtss2011.html#KennaHBMA11

Koshy T (2008) Catalan Numbers with Applications. Oxford University Press

Megel T, Sirdey R, David V (2010) Minimizing task preemptions and migrations in multiprocessor optimal real-time schedules. 2013 IEEE 34th Real-Time Systems Symposium 0:37–46, DOI http://doi.ieeecomputersociety.org/10.1109/RTSS.2010.22

Mills A, Anderson J (2010) A stochastic framework for multiprocessor soft real-time scheduling. In: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE, pp 311–320, DOI 10.1109/RTAS.2010.33

Regnier P, Lima G, Massa E, Levin G, Brandt SA (2011) Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In: RTSS, pp 104–115

RTSIM (2011) Real-time system simulator (rtsim). URL http://rtsim.sssup.it

SchedCAT (2014) The schedulability test collection and toolkit. URL https://github.com/brandenburg/schedcat/

Valente P (2014) Using a lag balance property to compute a tighter tardiness bound for global edf. Technical Report URL `http://algogroup.unimore.it/people/paolo/harm-bound/harm-extended.pdf`

Valente P, Lipari G (2005a) An upper bound to the lateness of edf on multiprocessors. Technical Report URL `http://algogroup.unimore.it/people/paolo/rtss05/edf_highb.pdf`

Valente P, Lipari G (2005b) An upper bound to the lateness of soft real-time tasks scheduled by edf on multiprocessors. IEEE 26th Real-Time Systems Symposium 0:311–320, DOI http://doi.ieeecomputersociety.org/10.1109/RTSS.2005.8

Ward BC, Erickson JP, Anderson JH (2013) A linear model for setting priority points in soft real-time systems. In: Proceedings of Real-Time Systems: The Past, the Present, and the Future, pp 192–205

A Proof of Lemma 28

For the reader's convenience, we repeat here the statement of Lemma 28. The following inequality holds:

$$\forall Q \in \{0,1,\ldots,G\} \quad \sum_{g=1}^{Q-1} \mathrm{lag}_g(s_g,s_Q) - M_{Q+1} \sum_{g=1}^{Q} U_g \frac{\sum_{v=1}^{g-1} \mathrm{lag}_v(s_v,s_g)}{M_g M_{g+1}} \leq 0. \quad (112)$$

Proof (Lemma 28) Computing the difference between the two terms in the LHS of (112) is not straightforward, because only few pairs of lag variations happen to regard the same task and time interval. We can however note that every lag variation is with respect to a time interval that does range between two chain start times for the tasks in $\hat{\tau}$. Any such time interval can be further decomposed into a sequence of consecutive sub-intervals, each ranging between two consecutive chain start times for the tasks in $\hat{\tau}$. That is, by (2), the argument of the first sum in the LHS of (112) can be rewritten as $\mathrm{lag}_g(s_g,s_{g+1}) + \mathrm{lag}_g(s_{g+1},s_{g+2}) + \ldots + \mathrm{lag}_g(s_{Q-1},s_Q)$, whereas the argument of the sum in the numerator of the second term can be rewritten as $\mathrm{lag}_v(s_v,s_{v+1}) + \mathrm{lag}_v(s_{v+1},s_{v+2}) + \ldots + \mathrm{lag}_v(s_{g-1},s_g)$. To take advantage of this similar decomposition, first we expand the sum $\sum_{g=1}^{Q-1} \mathrm{lag}_g(s_g,s_Q)$ as follows:

$$\sum_{g=1}^{Q-1} \mathrm{lag}_g(s_g,s_Q) \overset{(2)}{=} \mathrm{lag}_1(s_1,s_Q) + \mathrm{lag}_2(s_2,s_Q) + \ldots + \mathrm{lag}_{Q-1}(s_{Q-1},s_Q). \quad (113)$$

Then we decompose each term in the RHS of (113) into a sum of basic lag variations, and finally we sum all the lag variations across the same time interval among each other:

$$\begin{array}{lllll} \mathrm{lag}_1(s_1,s_2) + & \mathrm{lag}_1(s_2,s_3) & + \ldots + & \mathrm{lag}_1(s_{Q-1},s_Q) & + \\ & \mathrm{lag}_2(s_2,s_3) & + \ldots + & \mathrm{lag}_2(s_{Q-1},s_Q) & + \\ & & \vdots + & \vdots & \\ & & + & \mathrm{lag}_{Q-1}(s_{Q-1},s_Q) & = \\ \multicolumn{5}{l}{\mathrm{lag}_1(s_1,s_2) + \sum_{p=1}^{2} \mathrm{lag}_p(s_2,s_3) + \ldots + \sum_{p=1}^{Q-1} \mathrm{lag}_p(s_{Q-1},s_Q).} \end{array} \quad (114)$$

Combining the terms in the last line into a summation, and substituting the result in (113), we get

$$\sum_{g=1}^{Q-1} \mathrm{lag}_g(s_g,s_Q) = \sum_{g=2}^{Q} \sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1},s_g). \quad (115)$$

We decompose now the sum $\sum_{v=1}^{g-1} \mathrm{lag}_v(s_v, s_g)$ in (112). Replacing, syntactically, g with v and Q with g in (115), we get

$$\sum_{v=1}^{g-1} \mathrm{lag}_v(s_v, s_g) = \sum_{v=2}^{g} \sum_{p=1}^{v-1} \mathrm{lag}_p(s_{v-1}, s_v). \tag{116}$$

This equality allows us to decompose the factor $\sum_{g=1}^{Q} U_g \frac{\sum_{v=1}^{g-1} \mathrm{lag}_v(s_v,s_g)}{M_g M_{g+1}}$ in the second term of the LHS of (112). In particular, in the following equalities, first we apply (116), and then we expand the external sum (the first term resulting from the expansion is equal to 0, because $\sum_2^1 x = 0$ for any expression x):

$$\begin{aligned} \sum_{g=1}^{Q} U_g \frac{\sum_{v=1}^{g-1} \mathrm{lag}_v(s_v, s_g)}{M_g M_{g+1}} \overset{(116)}{=} \sum_{g=1}^{Q} U_g \frac{\sum_{v=2}^{g} \sum_{p=1}^{v-1} \mathrm{lag}_p(s_{v-1}, s_v)}{M_g M_{g+1}} = \\ 0 + U_2 \frac{\sum_{v=2}^{2} \sum_{p=1}^{v-1} \mathrm{lag}_p(s_{v-1}, s_v)}{M_2 M_3} + \ldots + U_Q \frac{\sum_{v=2}^{Q} \sum_{p=1}^{v-1} \mathrm{lag}_p(s_{v-1}, s_v)}{M_Q M_{Q+1}}. \end{aligned} \tag{117}$$

We expand now the external sum for each term of the last line, and, similarly to what we did in (114), sum all the lag variations across the same time interval with each other:

$$\begin{array}{ccccccc} U_2 \frac{\mathrm{lag}_1(s_1,s_2)}{M_2 M_3} & + & & & & & \\ U_3 \frac{\mathrm{lag}_1(s_1,s_2)}{M_3 M_4} & + & U_3 \frac{\sum_{p=1}^{2} \mathrm{lag}_p(s_2,s_3)}{M_3 M_4} & + & & & \\ \vdots & + & \vdots & + & \vdots & & \\ U_Q \frac{\mathrm{lag}_1(s_1,s_2)}{M_Q M_{Q+1}} & + & U_Q \frac{\sum_{p=1}^{2} \mathrm{lag}_p(s_2,s_3)}{M_Q M_{Q+1}} & + \ldots + & U_Q \frac{\sum_{p=1}^{Q-1} \mathrm{lag}_p(s_{Q-1},s_Q)}{M_Q M_{Q+1}} & = & \\ \sum_{v=2}^{Q} U_v \frac{\mathrm{lag}_1(s_1,s_2)}{M_v M_{v+1}} & + & \sum_{v=3}^{Q} U_v \frac{\sum_{p=1}^{2} \mathrm{lag}_p(s_2,s_3)}{M_v M_{v+1}} & + \ldots + & U_Q \frac{\sum_{p=1}^{Q-1} \mathrm{lag}_p(s_{Q-1},s_Q)}{M_Q M_{Q+1}}. & & \end{array} \tag{118}$$

Taking out the factors that do not depend on the indexes of the outer summations, and replacing the result in (117), we get

$$\begin{aligned} \sum_{g=1}^{Q} U_g \frac{\sum_{v=1}^{g-1} \mathrm{lag}_v(s_v,s_g)}{M_g M_{g+1}} = \\ \mathrm{lag}_1(s_1,s_2) \sum_{v=2}^{Q} \frac{U_v}{M_v M_{v+1}} + \left(\sum_{p=1}^{2} \mathrm{lag}_p(s_2,s_3)\right) \sum_{v=3}^{Q} \frac{U_v}{M_v M_{v+1}} + \ldots + \\ + \left(\sum_{p=1}^{Q-1} \mathrm{lag}_p(s_{Q-1},s_Q)\right) \frac{U_Q}{M_Q M_{Q+1}} = \\ \sum_{g=2}^{Q} \left[\left(\sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1},s_g)\right) \sum_{v=g}^{Q} \frac{U_v}{M_v M_{v+1}} \right]. \end{aligned} \tag{119}$$

Substituting (119) and (115) in the LHS of (112), we get

$$\begin{aligned} \sum_{g=1}^{Q-1} \mathrm{lag}_g(s_g, s_Q) - M_{Q+1} \sum_{g=1}^{Q} U_g \frac{\sum_{v=1}^{g-1} \mathrm{lag}_v(s_v,s_g)}{M_g M_{g+1}} &\overset{(119)+(115)}{=} \\ \sum_{g=2}^{Q} \sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1}, s_g) - M_{Q+1} \sum_{g=2}^{Q} \left[\left(\sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1},s_g)\right) \sum_{v=g}^{Q} \frac{U_v}{M_v M_{v+1}} \right] &= \\ \sum_{g=2}^{Q} \left\{ \sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1}, s_g) - M_{Q+1} \left(\sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1},s_g)\right) \sum_{v=g}^{Q} \frac{U_v}{M_v M_{v+1}} \right\} &= \\ \sum_{g=2}^{Q} \left[\left(\sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1},s_g)\right) \left(1 - M_{Q+1} \sum_{v=g}^{Q} \frac{U_v}{M_v M_{v+1}}\right) \right] &\overset{(87)}{=} \\ \sum_{g=2}^{Q} \left[\left(\sum_{p=1}^{g-1} \mathrm{lag}_p(s_{g-1},s_g)\right) \frac{M_{Q+1}}{M_g} \right] &\leq 0, \end{aligned} \tag{120}$$

where the last inequality follows from that all the factors $\sum_{p=1}^{g-1} \text{lag}_p(s_{g-1}, s_g)$ are lower than or equal to 0 by Lemma 21 with $v = p$, $t_p = s_{g-1}$ and $\bar{t} = s_g$, and that all the factors M_{Q+1}/M_g are positive by (86). □