An Upper Bound to the Lateness of Soft Real-time Tasks Scheduled by EDF on Multiprocessors*

Paolo Valente
Scuola Superiore S. Anna, Italy
pv@gandalf.sssup.it

Giuseppe Lipari
Scuola Superiore S. Anna, Italy
lipari@sssup.it

Abstract

Multiprocessors are now commonplace for efficiently achieving high computational power, even in embedded systems. A considerable research effort is being addressed to schedulability analysis of global scheduling in Symmetric Multiprocessor Platforms (SMP), where there is a global queue of ready tasks, and preemption and migration are allowed.

In many soft real-time applications (as e.g. multimedia and telecommunication) a bounded lateness is often tolerated. Unfortunately, when considering priority-driven scheduling of periodic/sporadic tasks, previous results only focused on guaranteeing all deadlines, and provided worst-case utilization bounds that are lower than the maximum available computational power. In particular, until now, the existence of an upper bound on the lateness of soft real-time tasks for a fully utilized SMP was still an open problem.

In this paper we do solve this problem by providing an upper bound to the lateness of periodic/sporadic tasks – with relative deadlines equal to periods/minimum inter-arrival times – scheduled by EDF on a SMP, under the only assumption that the total utilization is no higher than the total system capacity.

1. Introduction

Multiprocessors are now commonplace in general-purpose as well as in embedded systems. They provide a cost-effective solution to achieve high computational power. Besides, due to technological and physical constraints, increasing the speed of single processors is becoming more and more difficult. Hence multiprocessor platforms seem to be the only option for the most computationally demanding applications.

In the last year a large number of multi-core chips as well as multiprocessor architectures have been launched in the market. For example, to meet the requirements of demanding embedded real-time applications, ARM proposes MPCore, a synthesizable multiprocessor core, while Motorola proposes its PowerPMC-280 SMP platform. In the high-end general purpose processor market, both Intel, with its Pentium D brand, and AMD, with e.g. the Opteron dual-core processor, envision multi-core processors as the architecture of choice for high performance applications.

In this paper we consider soft real-time tasks to be executed on a Symmetric Multi Processor (SMP) platforms, comprised of M identical processors with constant speed. Unfortunately, multiprocessor platforms pose greater difficulties than single processor ones when applications have time requirements. Many negative results are known on the scheduling of real-time applications on multiprocessors, including SMPs [1, 2, 4, 8, 3, 7, 6, 12].

The results presented in this paper are related to the class of soft real-time applications that can be modeled as a set of periodic/sporadic tasks, i.e. sequences of jobs to execute, where each job is associated with a relative completion deadline equal to the period/minimum inter-arrival time. In soft real-time applications, deadlines are not critical, but it is important to respect some *Quality of Service* (QoS) requirements. Examples of such QoS constraints are: limited number of deadline misses, limited deadline miss percentage, and so on.

In this paper we are interested in soft real-time applications that can tolerate a bounded lateness with respect to the desired deadline. This kind of constraint matches a large class of applications, like multimedia, telecommunication, and financial ones. As an example, consider a video player: a given frame-rate must be guaranteed, but a jitter of few milliseconds in the frame-time does not significantly affect the quality of the video. In contrast, audio quality is extremely sensitive to silence gaps. However, audio samples are typically buffered and played back at the desired rate by the audio device. A bounded lateness in providing new samples to the device can be easily compensated using a pre-buffering strategy.

1.1. Related work

Research on real-time multiprocessor scheduling has been mainly focused on guaranteeing strict deadline observance. The two main approaches are *partitioning* and *global*

*This work has been supported in part by the European Commission under contract IST 2001-34820 (ARTIST project).

Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)
0-7695-2490-7/05 $20.00 © 2005 IEEE

scheduling. In partitioning the task set is divided – partitioned – into M groups. Each group of tasks is assigned to one of the processors, and processors are scheduled independently. The main advantage of such an approach is its simplicity, as a multiprocessor scheduling problem is reduced to M uniprocessor ones. Furthermore, since there is no migration, this approach presents a low overhead.

Unfortunately, there are various negative drawbacks. First, finding an optimal assignment of tasks to processors is a bin-packing problem, which is NP-hard in the strong sense. Hence, sub-optimal heuristics are usually adopted [13, 11, 9]. Second, there are task sets that are schedulable only if tasks are not partitioned [6]. Also, when tasks are allowed to dynamically enter and leave the system, a global re-assignment of tasks to processors may be necessary to balance the load, otherwise the overall utilization may decrease dramatically.

In global scheduling, jobs are inserted in a global priority-ordered *ready queue*, and at each time instant the available processors are allocated to the highest priority jobs in the ready queue. Tasks are in general subject to migration, i.e. during the system lifetime they may be executed on different processors.

An important classification is whether a scheduling algorithm is *priority-driven* [8], i.e. each job is assigned a fixed priority, or the priority of a job can vary over time. An important class of global schedulers of the second type is the class of PFair schedulers [5, 14]. PFair schedulers break jobs into smaller uniform pieces, which are then scheduled.

Unfortunately, in case of either partitioning or priority-driven scheduling, meeting all deadlines is paid in terms of schedulable utilization: any possible priority-driven and/or partitioned scheduling algorithm has a total worst-case utilization upper bound no larger than $\frac{M+1}{2}$ [6]. On the contrary, PFair algorithms are the only known schedulers able to meet all the deadlines still achieving full utilization. Unfortunately they may suffer from high scheduling and migration overhead.

1.2. Motivation

Until now, soft real-time applications could be scheduled on multiprocessor platforms either using efficient priority-driven schedulers and obtaining zero lateness, but wasting up to half of the available computational power; or using PFair algorithms, which do achieve full utilization with zero lateness, but may cause high overhead.

Except for PFair scheduling, to the best of the authors' knowledge, no lateness bound is available for soft real-time tasks that fully utilize a multiprocessor. In particular, it was not even known *if* lateness was actually bounded.

When considering partitioning, it is impossible to reach full utilization with bounded lateness, as shown by the following example. Consider 2 processors and 3 tasks, each one with utilization $2/3$. There is no way to assign all tasks to the processors and achieve bounded lateness. In fact, either we overload one of the processors, or we discard one of the tasks, achieving a total utilization of $4/3$.

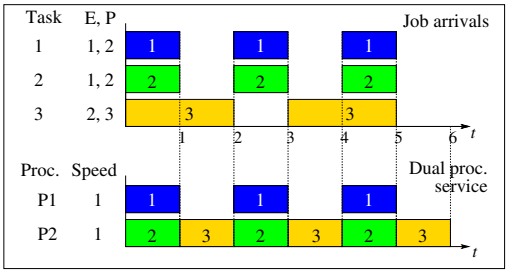

Figure 1. Example of unbounded lateness with fixed priority scheduling.

When considering global scheduling, not all priority-driven scheduling algorithms can achieve bounded lateness. Consider a system with 2 processors and 3 tasks, scheduled by fixed priority with priority assigned according to Rate Monotonic. Task 1 and 2 have computation time 1 and period 2; task 3 has computation time 2 and period 3. The total utilization is $5/3 < 2$. Job arrivals are shown in the top part of Fig. 1. Each arriving job is depicted as a rectangle: the projection of the left corner of each rectangle represents the arrival time of the corresponding job, while the length of the base is equal to the execution time of the job. The number on each rectangle refers to the task that issued the job. The schedule of the first 6 instants of time is shown in the bottom part of the figure. Notice that task 3 starts accumulating instances, and the lateness of each instance indefinitely increases.

In the previous example, it is easy to see that EDF would have not suffered from the problem of unbounded lateness, because jobs whose deadline is in the past have larger priority than newly arriving jobs. Intuitively, this property of EDF priorities apparently guarantees a bounded lateness. However, until now, providing an upper bound to the lateness of soft real-time tasks for a fully utilized SMP, and under priority-driven scheduling, was still an open problem.

1.3. Contribution

In this paper we consider a class of global priority-driven schedulers, the *DPS Finish Time Schedulers* (see Section 4 for a definition of this class), which EDF belongs to. We prove that these schedulers guarantee bounded lateness even when the system is fully utilized. We achieve this result by actually computing an upper bound to the maximum lateness in a simple closed form.

Is this bound tight? We performed a large number of simulation experiments to see how the actual maximum lateness experienced by the tasks compares to our worst-case bound. The bound resulted virtually tight in case of 2 processors: the ratio between the measured maximum lateness and the bound is 0.99. This ratio decreases as the number of processors increases, until it stabilizes at approximately $1/3$ for a number of processors higher than 10. All the results are discussed

Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)
0-7695-2490-7/05 $20.00 © 2005 IEEE

more extensively in Section 5.

The paper is organized as follows. In Section 2 we formally introduce the system and the notations. In Section 3 we present the main results, whereas in Section 4 we present the proofs. Finally, we report simulation results in Section 5.

2. System description and notations

We consider a system consisting of N periodic or sporadic tasks to be executed on a multiprocessor platform with M identical processors. All the processors have the same *speed* (*capacity*) R, measured in number of execution cycles per time units. Each task i consists of an infinite sequence of jobs J_i^j $j = 1, \ldots$ to be executed. Each job J_i^j is characterized by an activation (arrival) time a_i^j, a length $L(J_i^j)$, equal to the number of execution cycles for completing the job, and a completion deadline d_i^j. We say that a job J_i^j has an execution time $e_i^j \equiv \frac{L(J_i^j)}{R}$. The following relations hold:

$$\begin{aligned} a_i^j &\geq a_i^{j-1} + T_i \\ d_i^j &= a_i^j + T_i \end{aligned}$$

where T_i is the task period (minimum inter-arrival time). The completion (finish) time of the job J_i^j is denoted as f_i^j. We define as *lateness* of a job J_i^j the quantity $\mathrm{lat}_i^j \equiv \max\left[0,\, f_i^j - d_i^j\right]$.

We denote, respectively, with $L_i \equiv \max_j \left\{L(J_i^j)\right\}$ and $E_i \equiv \frac{L_i}{R}$ the worst-case job length and the worst-case job execution time for task i. Finally, we define $U_i \equiv \frac{E_i}{T_i} \leq 1$ as the *utilization* of task i. We assume that $\sum_{i=1}^{N} U_i \leq M$.

In [10] the concept of *predictable* scheduler is defined. A scheduler is predictable if, given two sets of jobs with the same cardinality and such that, for each job in the first set, there is a corresponding job in the second set with the same arrival time and priority, and with execution time no larger than the job in the first set, then the finish time of each job in the first set is no lower than the finish time of the corresponding job in the second set. They also proved that any priority driven scheduler is *predictable*. Hence, for simplicity, in the remainder we will assume that each job J_i^j has a length $L(J_i^j) = L_i$.

We assume that a job cannot start executing before the previous job of the same task has completed. We refer to this constraint as the *precedence constraint*. We stress the fact that a job can arrive also *before* the previous jobs of the same task have completed. We say that a job J_i^j is *pending* at time t if and only if $a_i^j \leq t < f_i^j$ (hence a job under service is still pending). Every task has a FIFO queue where its pending jobs are stored. We say that a task is *active* if it has pending jobs.

We define as *total speed* and *maximum total speed* of a multiprocessor at time t, respectively, $M_{\text{busy}}(t) \cdot R$ and $M \cdot R$, where $M_{\text{busy}}(t) \leq M$ is the number of busy processors at time t. We define as *under-load* and *full-load* periods the time intervals during which $M_{\text{busy}}(t) < M$ and $M_{\text{busy}}(t) = M$, respectively.

In the remainder of the paper we will refer to the above defined system as the *Multi Processor System* (MPS).

As stated in the introduction, we consider *global priority-driven* scheduling. At each time instant the available processors are allocated to the highest priority jobs in the ready queue. We assume that ties are arbitrarily broken. We allow preemption and migration, i.e. jobs can be suspended and later resumed on the same or on a different processor, due to the arrival of some higher priority job. In particular, we will focus on a special class of global priority-driven scheduling algorithms – defined in the next subsection – that includes EDF.

We call a *job fraction* any portion of a job continuously executed between two consecutive start (or resume) and suspend (or completion) events. We define as priority of a job fraction the priority of the job the fraction belongs to. We define a *chain* of jobs of a task any sequence of job fractions belonging to the same task and served back-to-back, and *head* of the chain the first job fraction in the sequence.

We will assume any generic function $f(t)$ of the time to be right continuous. Furthermore, for compactness, we set $f(x^-) = \lim_{t \to x^-} f(t)$, and we assume that the exponentiation a^x, with exponent $x = 0$, is always equal to 1 (even when the base a is infinite).

2.1. The Dedicated Processor System

In this subsection we introduce the *Dedicated Processor System* (DPS), a special reference system that we will use to define the class of global priority-driven schedulers for which our results hold.

Definition 1 *Given a MPS, we define as its reference* Dedicated Processor System *(DPS) the system consisting of the same task set and a multi-processor platform containing a dedicated processor for each of the N tasks in the MPS; each dedicated processor has a speed $R_i^{DPS} \equiv U_i \cdot R$ $i = 1, 2, \ldots, N$.*

For any job J_i^j we define as its *virtual finish time* the time instant F_i^j at which it is completed in the DPS. Since $T_i = \frac{E_i}{U_i}$ $\forall i$, and since we assumed that all the jobs of the i-th task have worst-case length L_i, we have that the DPS completes each job exactly on its deadline and, hence, no later than the arrival of the next job of the same task, i.e. $\forall J_i^j$ $F_i^j = d_i^j \leq a_i^{j+1}$. Consequently $\forall J_i^j$ $\mathrm{lat}_i^j = f_i^j - F_i^j$. This is the crucial property that we will exploit to compute an upper bound to the maximum lateness.

An example of the service provided by a MPS and by its reference DPS is shown in Fig. 2.A. The task set is comprised of 4 periodic tasks, all with period 4 and job length 3. Arriving jobs are depicted using the same conventions as in Fig. 1.

Jobs are scheduled by EDF in the MPS. Especially, since ties can be arbitrarily broken, in this example we chose to break ties in favor of lower index tasks to draw one of the

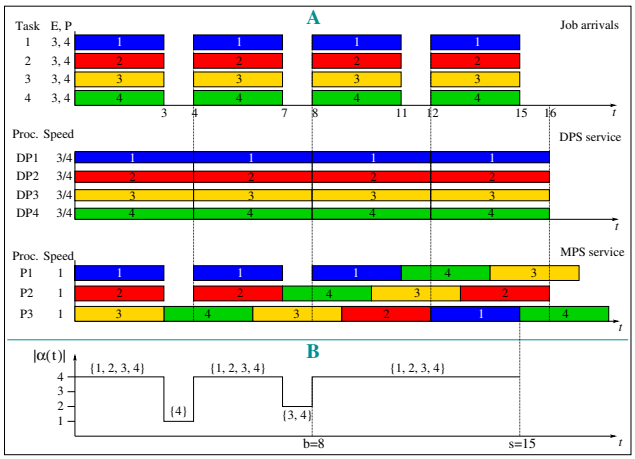

Figure 2. Comparing the MPS and the DPS.

possible schedules. The figure clearly shows that, whereas the DPS correctly schedules all the jobs, the MPS misses e.g. the deadline of job J_4^1 at time 4. Upon J_4^1 completion, task 4 has lateness 3. The situation gets worse during the second period, and both J_3^2 and J_4^2 miss their deadline at time 8.

Hereafter we will consider the following two systems: a generic MPS and its reference DPS. We will refer to these systems as *the* MPS and *the* DPS, respectively. We can now define the class of schedulers we will focus on.

Definition 2 *We say that a priority-driven scheduler for the MPS is a DPS Finish Time (DPS-FT) scheduler, if, denoted with P_i^j the priority of the generic job J_i^j, we have that*

$$\forall J_i^j,\ J_k^l \quad \begin{cases} P_i^j = P_k^l \iff F_i^j = F_k^l \\ P_i^j > P_k^l \iff F_i^j < F_k^l \end{cases} \tag{1}$$

and, at each time instant, the available processors are allocated to the highest priority jobs. Ties are arbitrarily broken.

In other words, in a DPS-FT scheduler the ordering among job priorities is the opposite of the ordering between job finish times in the DPS. Since $\forall J_i^j \ \ F_i^j = d_i^j$, EDF is a DPS-FT scheduler. Hereafter, we will assume that a DPS-FT scheduler is used to schedule jobs in the MPS.

Under the assumptions of constant speed processors and of tasks with constant job length, any DPS-FT scheduler is equivalent to EDF (i.e. it generates the same schedules). However, all the following lemmas and theorems will be actually proved in the more general case where all the processors have the same time-varying speed $R(t)$, and where each dedicated processor has time-varying speed $R_i^{DPS}(t) = U_i \cdot R(t)$. In this case, the class of DPS-FT schedulers can also include schedulers different from EDF. While this generalization does not complicate the proofs, it paves the way for future more general results.

We define as $W_i^{MPS}(t)$ and $W_i^{DPS}(t)$ the amount of service provided by, respectively, the MPS and the DPS to the i-th task during $[0, t]$. We define the total amount of service provided by the MPS and the DPS during $[0, t]$ as, respectively, $W^{MPS}(t) \equiv \sum_i W_i^{MPS}(t)$ and $W^{DPS}(t) \equiv \sum_i W_i^{DPS}(t)$. We define as *lag* of the i-th task at time t the following quantity:

$$\text{lag}_i(t) \equiv W_i^{DPS}(t) - W_i^{MPS}(t)$$

For brevity, given two time instants $t_2 > t_1$, we define $W_i^{MPS}(t_1, t_2) \equiv W_i^{MPS}(t_2) - W_i^{MPS}(t_1)$. We use the same short notation for W_i^{DPS}, W^{MPS}, W^{DPS} and lag_i.

In the proofs we will often use the following property: since $R_i^{DPS} \leq R\ \forall i$, the lag of a task can not increase during the service of one of its job chains. For example, in Fig. 2.A the lag of task 4 increases during $[0, 3]$, and it is equal to $\frac{9}{4}$ at time 3. Conversely, it decreases during $[3, 6]$, and it is e.g. equal to 2 at time 4.

Since the lag of a task may be a useful figure of merit, in this paper we report an upper bound to the maximum per-task lag in addition to the one on the maximum lateness. The notations introduced until now are summarized in Table 2.1.

R	Speed of any of the processors
M	Number of processors in the system
$W^S(t)$	Total amount of service delivered by the system S during $[0,\ t]$
$W_i^S(t)$	Amount of service received by the i-th task during $[0,\ t]$ in a system S
$L(J)$	Length (num. of execution cycles) of job J
J_i^j	The j-th job of the i-th task
a_i^j, s_i^j, f_i^j	Arrival time, start time, finish time of J_i^j
F_i^j	(Virtual) finish time of J_i^j in the DPS
L_i	(Worst-case) length of i-th task
E_i	(Worst-case) execution time of i-th task
L_{max}	Maximum job length over all the tasks
E_{max}	Maximum execution time over all the tasks
$\text{lag}_i(t)$	Lag of task i ($W_i^{DPS}(t) - W_i^{MPS}(t)$).

Table 1. Notations used in this paper.

3. Maximum lag and maximum lateness

In this section we enunciate and briefly discuss the following theorems, which constitute the main results of this paper.

Theorem 1 *If an MPS comprised of M identical processors is scheduled using a DPS-FT scheduler, the following guarantees on the lag experienced by any task hold:*

$$\forall i,\ t \quad \text{lag}_i(t) \leq (1 - \frac{U_i}{M}) \cdot L_i + U_i \cdot (\frac{M}{M-1})^{M-3} \cdot L_{max} \tag{2}$$

$$\forall J_i^j \ \text{lag}_i(f_i^j) \leq U_i \cdot \left[\frac{M-1}{M} \cdot L_i + (\frac{M}{M-1})^{M-3} \cdot L_{max} \right] \tag{3}$$

Theorem 2 *If an MPS comprised of M identical constant speed processors is scheduled using a DPS-FT scheduler, the following guarantees on the job lateness hold:*

$$\forall J_i^j \quad \text{lat}_i^j \leq \frac{M-1}{M} \cdot E_i + (\frac{M}{M-1})^{M-3} \cdot E_{max}. \quad (4)$$

The formal proofs of the theorems (and of the next corollary) are reported in the next section. We can note that processors are not required to have constant speed for Theorem 1 to hold (but they must be identical, i.e. they must all have the same speed at any time instant). With regard to Theorem 2, we highlight that, when $M = 1$, Eq. (4) collapses to the EDF guarantee $\forall J_i^j$ $\text{lat}_i^j = 0$, since $(\frac{M}{M-1})^{M-3}$ becomes equal to zero.

It is easy to prove that the right terms in Inequalities (2), (3) and (4) are all non-decreasing functions of M, and that $(\lim_{M\to+\infty} \frac{M}{M-1})^{M-1} = e$. It follows that:

Corollary 1 *If an MPS comprised of M identical processors is scheduled using a DPS-FT scheduler, the following inequalities hold:*

$$\forall i,\ t \quad \text{lag}_i(t) \leq (1 - \frac{U_i}{M}) \cdot L_i + e \cdot L_{max} \quad (5)$$

$$\forall J_i^j \quad \text{lag}_i(f_i^j) \leq U_i \cdot [L_i + e \cdot L_{max}] \quad (6)$$

Furthermore, if processors have constant speed, the following inequality holds:

$$\forall J_i^j \quad \text{lat}_i^j \leq E_i + e \cdot E_{max}. \quad (7)$$

As can be seen, the corollary provides simpler but more conservative upper bounds. Finally, it is worth noting that all the above results hold also when tasks *fully utilize* the system.

4. Proofs

In this section we will formally prove Theorems 1 and 2, and Corollary 1. First, we introduce the notations used in the proofs and the proof strategy. The proofs are essentially based on computing a bound to $\text{lag}_i(t)$, from which the bound on the lateness will be then derived.

The following Lemma restricts the time instants to be considered when computing an upper bound to the lag.

Lemma 1 *The maximum lag experienced by a task is no higher than the maximum lag that the task can experience at the start time of some of its job fractions.*

Proof. When a task is inactive, its lag is necessarily no higher than 0. Consider instead a generic maximal active period $[t_1,\ t_2]$ of task i. Let X_i^k be the k-th job fraction of task i served by the MPS. Any time instant $t \in [t_1,\ t_2]$ necessarily falls into a sub-interval $[f_i^{k-1},\ f_i^k] \subseteq [t_1,\ t_2]$ ranging from the finish time f_i^{k-1} of a job fraction X_i^{k-1}, and the finish time of the next job fraction X_i^k served by the MPS (if X_i^k is the first job fraction of task i executed during $[t_1,\ t_2]$, then we assume $f_i^{k-1} = t_1$). Since the lag can not increase during the service of a job, we have that

$$\max_{t\in[f^{k-1},\ f_i^k]} \text{lag}_i(t) = \text{lag}_i(s_i^k)$$

where s_i^k is the start time of the fraction X_i^k. □

Consequently, in the next subsection we will focus on computing the maximum lag of the task at the start time s of a generic job fraction X belonging to a job J_i^j.

First, if $s = a_i^j$, then $\text{lag}_i(s) = 0$ because both the MPS and DPS have finished all the pending jobs at s^-.

Let us then consider the case $s > a_i^j$ (note that, in general, s might be even larger than F_i^j, i.e. larger than the job deadline in case of EDF).

To handle this case, we define as $\alpha(t)$ the set of the tasks owning pending jobs with priority no lower than X at time t. For example, in case of EDF, $\alpha(t)$ includes all the tasks owning jobs with deadline no higher than J_i^j (i.e. than the job X belongs to) at time t. Note that $\forall t \in [a_i^j,\ s)\ \ i \in \alpha(t)$, because J_i^j is pending during $[a_i^j,\ s)$ and, by definition, its priority is equal to the priority of its fraction X. Fig. 2.B shows the values assumed by $\alpha(t)$ and $|\alpha(t)|$ during $[0,\ s)$, assuming the fraction X to coincide with the whole job J_4^4, which in turn starts service at time $s = 15$ in Fig. 2.A.

There are only two possible causes for X to start at time $s > a_i^j$: 1) X is *blocked by priority*, that is at least M tasks own pending jobs with priority no lower than J_i^j at time s^- ($|\alpha(s^-)| \geq M$); 2) X is blocked by the precedence constraint at time s^-.

In the second case, X belongs to a chain. Since the lag of a task does not increase while its jobs are being served, the maximum lag of the task is trivially upper bounded by its maximum lag at the start time of the chain head. This is in its turn equal to 0, unless the chain head is blocked by priority. As a conclusion, the problem that remains to solve is computing the maximum lag of the task at the start time of a fraction blocked by priority. To this aim, we will use the following two definitions.

Definition 3 *Given a job fraction X blocked by priority, we define as* last priority blocking period *for X the time interval $[b, s)$, where b is the smallest time instant b such that $\forall t \in [b, s)\ |\alpha(t)| \geq M$, i.e. such that at least M tasks are continuously active and have pending jobs with priority no lower than X during $[b, s)$.*

Fig. 2.B shows the last priority blocking period of the job J_4^4. We note that b might in general precede a_i^j. Furthermore, we will exploit the following two properties of the last priority blocking period: $|\alpha(b^-)| < M$, and the MPS is in full load during $[b,\ s)$.

Definition 4 *We define Γ as the set of the jobs that receive service in the MPS during $[b,\ s)$.*

Notice that, by definition of last priority blocking period, the jobs in Γ have priority no lower than X. As an example, assuming again $X = J_4^4$ in Fig. 2, we have that $\Gamma = \{J_3^2,\ J_4^2,\ J_1^3,\ J_2^3,\ J_3^3,\ J_4^3,\ J_1^4,\ J_2^4,\ J_3^4\}$.

The MPS starts serving X only after serving part of the jobs in Γ. However, the jobs in Γ have priority no lower than

J_i^j, which means that they finish no later than J_i^j in the DPS. Hence, the DPS must complete *all* the jobs in Γ before it can complete J_i^j. Furthermore, since the MPS works at maximum total speed during $[b, s]$, it *consumes* the jobs in Γ at a pace no lower than the one at which the DPS could consume them during the same time interval. For these reasons, intuitively, the maximum value of $\mathrm{lag_i(s)}$ depends on *how ahead* is the DPS with respect to the MPS in the service of the jobs in Γ at time b. More formally, we will show that the maximum value of $\mathrm{lag_i(s)}$ depends on the following quantity: $\sum_{j\in\alpha^{pos}} \mathrm{lag}_j(b)$, where $\alpha^{pos} \subseteq \alpha(b)$ is the subset of the tasks with positive lag at time b. We call *total lag* related to the fraction X the above quantity.

We can now define the proof strategy: we will first express the maximum lag of a task as a function of the total lag in Subsection 4.1. This general formula will serve two purposes. We will first use it in Subsection 4.2 to compute an upper bound to the total lag itself. Then, in the last subsection, we will substitute the just computed bound in the general formula, thus getting an upper bound to the lag experienced by a task. Finally, the latter bound will be used to compute an upper bound to the lateness experienced by a job. A more detailed version of any of the following proofs can be found in [15].

4.1. Basic lemmas

This subsection contains four lemmas: the first two lemmas allows us to provide an upper bound to the lag of a task as a function of the total lag, whereas the last two lemmas are just algebraic facilities that will be used in next subsection to compute the maximum total lag.

As an intermediate step for computing an upper bound to the maximum lag of a task, the next lemma provides an upper bound to the difference between the total amount of service $W^{MPS}(b,\, s)$ that the MPS provides during $[b,\, s]$ and the total amount of service $W^{DPS}(b,\, F_i^j)$ that the DPS must provide during $[b,\, F_i^j]$ to finish J_i^j.

In the lemma, a special set of tasks σ is defined. We will discuss the use of σ just after enunciating the lemma.

Lemma 2 *Let X be a generic job fraction, belonging to a job J_i^j, that starts service at time s in the MPS after being blocked by priority. Let σ be any subset of the $M-1$ tasks whose jobs are under execution at time s, excluding task i. We have:*

$$W^{MPS}(b,\, s) - W^{DPS}(b,\, F_i^j) \;\leq\; \sum_{h\in\alpha^{pos}} \mathrm{lag}_h(b) - \sum_{h\in\sigma} \mathrm{lag}_h(s) - L^{res}(J_i^j) \quad (8)$$

where $L^{res}(J_i^j)$ is the difference between the length of J_i^j and the portion of J_i^j already served by the MPS at time s, b is the beginning of the last priority blocking period of X, and $\alpha^{pos} \equiv \{i \in \alpha(b) \,|\, \mathrm{lag}_i(b) > 0\}$.

Before the proof, a quick comment on the set σ. It can be any subset of the tasks that are active at time s, i excluded. For example, assuming $X = J_4^4$ in Fig. 2, the possible values of σ are $\{2\}$, $\{3\}$ and $\{2,\, 3\}$. The lemma holds for any choice of σ. In the following, we will use this same lemma with different values of σ to achieve different results. In fact, if we set $\sigma = \emptyset$, Inequality (8) provides an upper bound to $W^{MPS}(b,\, s) - W^{DPS}(b,\, F_i^j)$ as a function of the total lag, which will be used for computing an upper bound to the lag of any task. Conversely, the case $\sigma \neq \emptyset$ will be used when computing an upper bound to the total lag. The proof of the lemma follows.

Proof. To prove Inequality (8), we will first compute an upper bound to $W^{MPS}(b,\, s)$, then a lower bound to $W^{DPS}(b,\, F_i^j)$, and finally subtract them. Let $\Gamma(\sigma) \subseteq \Gamma$ be the subset of the jobs in Γ issued by the tasks in σ, and $\Gamma(\overline{\sigma}) \equiv \Gamma \backslash \Gamma(\sigma)$. For both bounds, we will separate the contribution due to the jobs in $\Gamma(\sigma)$ from the contribution due to the jobs in $\Gamma(\overline{\sigma})$. We start by computing an upper bound to $W^{MPS}(b,\, s)$.

During $[b, s)$ the MPS serves only some fractions (at most all the fractions) of the jobs in the previous two sets. Hence, defined as $L(\Gamma(\overline{\sigma}))$, $W^{MPS}_{\Gamma(\overline{\sigma})}(b)$ and $W^{MPS}_{\Gamma(\sigma)}(b, s)$ the sum of the length of the jobs in $\Gamma(\overline{\sigma})$, the service that the MPS gave to the jobs in $\Gamma(\overline{\sigma})$ before time b and the amount of service provided to the tasks in σ by the MPS during $[b, s]$, respectively, we have that:

$$W^{MPS}(b,\, s) \leq L(\Gamma(\overline{\sigma})) - W^{MPS}_{\Gamma(\overline{\sigma})}(b) + W^{MPS}_{\Gamma(\sigma)}(b, s) \quad (9)$$

Second, we compute a lower bound for $W^{DPS}(b,\, F_i^j)$. Observe that: 1) not all the jobs in Γ have arrived at time b, 2) the DPS must complete all the jobs in Γ no later than F_i^j. It follows that $F_i^j \geq b$. As done before, we separate the contribution of the jobs in $\Gamma(\sigma)$ from the one of the jobs in $\Gamma(\overline{\sigma})$. With regard to the latter set, the DPS must certainly have served both all the jobs in $\Gamma(\overline{\sigma})$, and the portion $L^{res}(J_i^j)$ before finishing J_i^j. However, part of these jobs might have arrived before b, and the DPS might have already (partially) served them. The service provided to these jobs can be written as $W^{DPS}_{\Gamma(\overline{\sigma})}(b) + W^{DPS}_{J_i^j}(b)$. In the end, we can write:

$$\begin{aligned} W^{DPS}(b, F_j^j) \;\geq\; & \\ L(\Gamma(\overline{\sigma})) + W^{DPS}_{\Gamma(\sigma)}(b, F_i^j) + L^{res}(J_i^j) + & \\ - (W^{DPS}_{\Gamma(\overline{\sigma})}(b) + W^{DPS}_{J_i^j}(b)). & \end{aligned} \quad (10)$$

Now we find an upper bound to the last term. Consider a generic job $J \in \Gamma(\overline{\sigma})$. If at time b the DPS has already served a portion of J larger than the portion already served by the MPS, then J is pending in the MPS at time b and has priority no lower than X. Let q be the task that generated J. We have that $\mathrm{lag}_q(b) > 0$. Let $W_J^{MPS}(b)$ and $W_J^{DPS}(b)$ be the service that the MPS and DPS give to J before b, respectively. We have that $W_J^{DPS}(b) > W_J^{MPS}(b) \geq 0$. Finally, let $W_{\overline{J}}^{MPS}(b)$ and $W_{\overline{J}}^{DPS}(b)$ be, respectively, the service that the MPS and the DPS give to the jobs of q, excluding J, before b.

By definition, $q \in \alpha^{pos} \backslash \sigma$. Moreover, from the definition of $\mathrm{lag}_q(b)$ we get $W_J^{DPS}(b) = \mathrm{lag}_q(b) + W_J^{MPS}(b) - (W_{\overline{J}}^{DPS}(b) - W_{\overline{J}}^{MPS}(b))$. Since $W_J^{DPS}(b) > W_J^{MPS}(b)$,

$W_{\overline{J}}^{DPS}(b) - W_{\overline{J}}^{MPS}(b) \geq 0$. In the end,

$$W_J^{DPS}(b) \leq \text{lag}_q(b) + W_J^{MPS}(b).$$

Summing over all the jobs $J \in \Gamma(\overline{\sigma})$ and applying the above arguments to $W_{J_i^j}^{DPS}(b)$ as well, we get:

$$\begin{array}{rcl} W_{\Gamma(\overline{\sigma})}^{DPS}(b) + W_{J_i^j}^{DPS}(b) & \leq & \\ \sum_{q\in\alpha^{pos}\setminus\sigma} \text{lag}_q(b) + W_{\Gamma(\overline{\sigma})}^{MPS}(b) & \leq & \\ \sum_{q\in\alpha^{pos}} \text{lag}_q(b) + W_{\Gamma(\overline{\sigma})}^{MPS}(b) & & \end{array} \quad (11)$$

We can divide σ into two subset, a subset $\sigma_1 \subseteq \alpha(b)$ and a subset σ_2 such that $\sigma_2 \cap \alpha(b) = \emptyset$. Finally, defined $\sigma_1^{pos} \subseteq \sigma_1$ as the subset of the tasks whose lag is positive at time b, we have

$$\begin{array}{rcl} \sum_{q\in\alpha^{pos}\setminus\sigma} \text{lag}_q(b) & = & \\ \sum_{q\in\alpha^{pos}} \text{lag}_q(b) - \sum_{q\in\sigma_1^{pos}} \text{lag}_q(b) & \leq & \\ \sum_{q\in\alpha^{pos}} \text{lag}_q(b) - \sum_{q\in\sigma_1} \text{lag}_q(b) & = & \\ \sum_{q\in\alpha^{pos}} \text{lag}_q(b) - \sum_{q\in\sigma} \text{lag}_q(b). & & \end{array} \quad (12)$$

where the last equality follows from the fact that $\forall q \in \sigma_2\ \text{lag}_q(b) = 0$.

Substituting (12) in (11), we get

$$W_{\Gamma(\overline{\sigma})}^{DPS}(b) + W_{J_i^j}^{DPS}(b) \leq \sum_{q\in\alpha^{pos}} \text{lag}_q(b) - \sum_{q\in\sigma} \text{lag}_q(b). \quad (13)$$

At this point, we can return back to Inequality (10) and write:

$$\begin{array}{rcl} W^{DPS}(b, F_j^j) & \geq & \\ L(\Gamma(\overline{\sigma})) + W_{\Gamma(\sigma)}^{DPS}(b, F_i^j) + L^{res}(J_i^j) + & & \\ - \sum_{q\in\alpha^{pos}} \text{lag}_q(b) + \sum_{q\in\sigma} \text{lag}_q(b) - W_{\Gamma(\overline{\sigma})}^{MPS}(b) & & \end{array}$$

Subtracting the last inequality from (9), we get:

$$\begin{array}{rcl} W^{MPS}(b,\, s) - W^{DPS}(b,\, F_i^j) & \leq & \\ W_{\Gamma(\sigma)}^{MPS}(b) - W_{\Gamma(\sigma)}^{DPS}(b, F_i^j) + & & \\ -L^{res}(J_i^j) + \sum_{q\in\alpha^{pos}} \text{lag}_q(b) - \sum_{q\in\sigma} \text{lag}_q(b). & & \end{array} \quad (14)$$

We can simplify the above expression by considering that

$$\begin{array}{rcl} W_{\Gamma(\sigma)}^{MPS}(b,\, s) - W_{\Gamma(\sigma)}^{DPS}(b,\, F_i^j) & \leq & \\ W_{\Gamma(\sigma)}^{MPS}(b,\, s) - W_{\Gamma(\sigma)}^{DPS}(b,\, s) & = & \\ - \sum_{j\in\sigma} \text{lag}_j(b,\, s) & & \end{array} \quad (15)$$

Substituting (15) in (14), we get the thesis.

□

Using the bound computed in the previous lemma, we can now prove the following lemma, which expresses the maximum lag of a task as a function of the total lag.

Lemma 3 *Let X be a generic job fraction, belonging to a job J_i^j, that starts service at time s in the MPS after being blocked by priority. Let σ be any subset of the $M-1$ tasks , excluding task i, whose jobs are under execution at time s. We have:*

$$\begin{array}{rcl} \text{lag}_i(s) \leq L^{res}(J_i^j) + & & \\ + \frac{U_i}{M} \cdot \left[\sum_{j\in\alpha^{pos}} \text{lag}_j(b) - \sum_{j\in\sigma} \text{lag}_j(s) - L^{res}(J_i^j) \right] & & \end{array} \quad (16)$$

where $L^{res}(J_i^j)$ is the difference between the length of J_i^j and the portion of J_i^j already served by the MPS at time s, b is the beginning of the last priority blocking period of X, and $\alpha^{pos} \equiv \{i \in \alpha(b) \,|\, \text{lag}_i(b) > 0\}$.

Proof. The proof strategy is as follows: we will first express $\text{lag}_i(F_i^j)$ in a convenient form, then we will find an upper bound to $\text{lag}_i(s) - \text{lag}_i(F_i^j)$, finally we will sum this bound to $\text{lag}_i(F_i^j)$. We have that:

$$W_i^{DPS}(F_i^j) = W_i^{MPS}(s) + L^{res}(J_i^j) \quad (17)$$

We can do some algebraic manipulations:

$$W_i^{MPS}(s) = W_i^{MPS}(F_i^j) + \Delta \quad (18)$$

where $\Delta \equiv W_i^{MPS}(s) - W_i^{MPS}(F_i^j)$. Substituting successively (17) and (18) in the definition of $\text{lag}_i(F_i^j)$, we get

$$\text{lag}_i(F_i^j) = L^{res}(J_i^j) + \Delta \quad (19)$$

We will now compute an upper bound to $\text{lag}_i(s) - \text{lag}_i(F_i^j)$. From (18) we get:

$$\text{lag}_i(s) - \text{lag}_i(F_i^j) = \left[W_i^{DPS}(s) - W_i^{DPS}(F_i^j) \right] - \Delta \quad (20)$$

We want now to exploit Lemma 2 to find an upper bound to the difference $W_i^{DPS}(s) - W_i^{DPS}(F_i^j)$ in (20). To this aim, we can first put this difference in a more convenient form:

$$W_i^{DPS}(s) - W_i^{DPS}(F_i^j) = W_i^{DPS}(b,\, s) - W_i^{DPS}(b,\, F_i^j) \quad (21)$$

We need a last step arrive to a form similar to the left member of Inequality (8). To this aim, we will work on $W_i^{DPS}(b,\, s)$. Since $[b,\, s]$ falls inside a full-load period (at least M tasks are active during $[b,\, s)$), then $\forall t \in [b,\, s]$ the total speed of the MPS at time t is $R^{MPS}(t) = M \cdot R(t)$. In contrast, the total speed $R^{DPS}(t)$ of the DPS is:

$$R^{DPS}(t) = \sum_{j\in A^{DPS}(t)} U_j \cdot R(t) = \frac{\sum_{j\in A^{DPS}(t)} U_j}{M} \cdot R^{MPS}(t)$$

where $A^{DPS}(t)$ is the set of the tasks active under the DPS at time t. Furthermore, $W_i^{DPS}(b,\, s)$ is maximum if the DPS continuously serves the *i-th* task during $[b,\, s]$. Defined $\chi_i(t)$ as the fraction of the total speed that the DPS dedicates to the $i-th$ task at time $t \in [b,\, s]$ under this hypothesis, we have $\chi_i(t) = \frac{U_i}{\sum_{j\in A^{DPS}(t)} U_j}$. In the end:

$$\begin{array}{rcl} W_i^{DPS}(b,\, s) & \leq & \\ \int_b^s \chi_i(\tau) \cdot R^{DPS}(\tau) \cdot d\tau & = & \\ \int_b^s \frac{U_i}{\sum_{j\in A^{DPS}(\tau)} U_j} \cdot \frac{\sum_{j\in A^{DPS}(\tau)} U_j}{M} \cdot R^{MPS}(\tau) & = & \\ \frac{U_i}{M} \cdot W^{MPS}(b,\, s) & & \end{array} \quad (22)$$

As a conclusion, substituting the last inequality in (21) and exploiting (8), we get

$$\begin{array}{rcl} W_i^{DPS}(s) - W_i^{DPS}(F_i^j) & \leq & \\ \frac{U_i}{M} \cdot \left[W^{MPS}(b,\, s) - W^{DPS}(b,\, F_i^j) \right] & & \\ \frac{U_i}{M} \cdot \left[\sum_{j\in\alpha^{pos}} \text{lag}_j(b) - \sum_{j\in\sigma} \text{lag}_j(s) - L^{res}(J_i^j) \right] & & \end{array}$$

Substituting the last inequality in (20), and summing to (19), we get the thesis. □

The following two purely algebraic Lemmas will prove useful in the next subsection. For space constraints, we removed their proofs, which can be found in [15].

Lemma 4 *Defined the following two functions:*

$$g_i(x) \equiv \begin{cases} x & i = 1 \\ A_i + E_i \cdot \left[B_i - \sum_{j=1}^{i-1} g_j(x)\right] & i = 2, \ldots, K \end{cases}$$

with $A_i \geq 0$, $B_i \geq 0$ *and* $0 \leq E_i \leq \frac{1}{M}$ $i = 2, \ldots, K$, *and* $1 \leq K \leq M$; *and defined*

$$f(K, A_2, \ldots, A_K, B_2, \ldots, B_K, E_2, \ldots, E_K, x) \equiv \sum_{i=1}^{K} g_i(x)$$

$f(K, A_2, \ldots, A_K, B_2, \ldots, B_K, E_2, \ldots, E_K, x)$ *is a non-decreasing function of* x.

Lemma 5 *We define*

$$h_i \equiv C + P_i \cdot \left[D - \sum_{j=1}^{i-1} h_j\right] \quad i = 1, 2, \ldots, K \tag{23}$$

with $C > 0$, $D > 0$,$0 \leq P_i \leq \frac{1}{M}$ $i = 2, \ldots, K$, $1 \leq K \leq M$, *and assuming* $\sum_{j=l}^{m} a_j \equiv 0$ if $l < m$. *We have*

$$z(K, C, D, P_1, P_2, \ldots, P_K) \equiv \sum_{i=1}^{K} h_i \leq K \cdot C + D \cdot \left[1 - (\tfrac{M-1}{M})^K\right]$$

4.2. Bounding the total lag

We need a last intermediate lemma.

Lemma 6 *Let* $A(\bar{t})$ *be any subset of* $V \leq M$ *tasks under service at time* $\bar{t}$. *We have that:*

$$\sum_{j \in A(\bar{t})} \mathrm{lag}_j(\bar{t}) \leq V \cdot \tfrac{M-1}{M} \cdot L_{max} + \left[1 - (\tfrac{M-1}{M})^V\right] \cdot \max_{j \in A(\bar{t})} \left[\sum_{i \in \alpha_j^{pos}} \mathrm{lag}_i(b_j)\right]$$

where b_j *is the beginning of the last priority blocking period of the head* X_j *of the chain, under service at time* $\bar{t}$, *of the* $j-th$ *task in* $A(\bar{t})$, α_j^{pos} *is the set of the tasks whose pending jobs have priority no lower than* X_j, *and whose lag is positive at time* b_j .

Proof. We will first find an upper bound to $\sum_{j \in A(\bar{t})} \mathrm{lag}_j(\bar{t})$ with the same form of function z in Lemma 5, then we will apply this lemma to prove the thesis. We can assume, without losing generality, that the V tasks in $A(\bar{t})$ are the tasks 1, 2, ..., V, and that they are ordered by the start time s_j of the chain heads X_j. Let $J(X_j)$ be the job the fraction X_j belongs to. We define

$$\mathrm{Lag} \equiv \max_{j \in A(\bar{t})} \left[\sum_{i \in \alpha_j^{pos}} \mathrm{lag}_i(b_j)\right]$$

From Lemma 3 and posing $\sigma = \{1, 2, \ldots, V-1\}$, we can write

$$\mathrm{lag}_V(\bar{t}) \leq \mathrm{lag}_V(s_V) \leq \tfrac{M-U_V}{M} \cdot L^{res}(J(X_V)) + \tfrac{U_V}{M} \cdot \left[\mathrm{Lag} - \sum_{j=1}^{V-1} \mathrm{lag}_j(s_V)\right] \leq \tfrac{M-U_V}{M} \cdot L_{max} + \tfrac{U_V}{M} \cdot \left[\mathrm{Lag} - \sum_{j=1}^{V-1} \mathrm{lag}_j(\bar{t})\right]$$

because the lag of a task can not increase during the execution of one of its chains (tasks 1, 2, ..., $V-1$ are continuously served during $[s_V, t]$). Now we define

$$\lambda_j(\bar{t}) \equiv \frac{M-1}{M} \cdot L_{max} + \frac{U_j}{M} \cdot \left[\mathrm{Lag} - \sum_{i=1}^{j-1} \mathrm{lag}_i(\bar{t})\right]$$

Considering that $\mathrm{lag}_V(\bar{t}) \leq \lambda_V(\bar{t})$ and adding $\mathrm{lag}_{V-1}(\bar{t})$ we get

$$\mathrm{lag}_{V-1}(\bar{t}) + \mathrm{lag}_V(\bar{t}) \leq \mathrm{lag}_{V-1}(\bar{t}) + \lambda_V(\bar{t}) = f(2, A_2 = \tfrac{M-1}{M} \cdot L_{max}, B_2 = \mathrm{Lag} - \sum_{i=1}^{V-2} \mathrm{lag}_i(\bar{t}), E_2 = \tfrac{U_V}{M}, \mathrm{lag}_{V-1}(\bar{t})) \tag{24}$$

where the function f is the one defined in Lemma 4, and, as such, it is a non-decreasing function of $\mathrm{lag}_{V-1}(\bar{t})$. Hence, to find an upper bound to its value, we look for the maximum value that can be assumed by $\mathrm{lag}_{V-1}(\bar{t})$.

From Lemma 3, posing $\sigma = \{1, 2, \ldots, V-2\}$, and repeating the same steps as above, we get $\mathrm{lag}_{V-1}(\bar{t}) \leq \lambda_{V-1}(\bar{t})$. Hence, considering (24), the previous bound on $\mathrm{lag}_{V-1}(\bar{t})$ and Lemma 4, we have

$$\mathrm{lag}_{V-1}(\bar{t}) + \mathrm{lag}_V(\bar{t}) \leq \lambda_{V-1}(\bar{t}) + \lambda_V(\bar{t})$$

By inductively applying this argument, after V steps we get:

$$\sum_{j=1}^{V} \mathrm{lag}_j(\bar{t}) \leq \sum_{j=1}^{V} \lambda_j(\bar{t})$$

Hence, the thesis follows from applying Lemma 5 to $\sum_{j=1}^{V} \lambda_j(\bar{t}) = z(V, C = \frac{M-1}{M} \cdot L_{max}, D = \mathrm{Lag}, P_1 = \frac{U_1}{M}, P_2 = \frac{U_2}{M}, \ldots, P_V = \frac{U_V}{M})$.
□

We can now compute an upper bound to the total lag.

Theorem 3 *For any job fraction* X *which starts service in the MPS after being blocked by priority, we have*

$$\sum_{j \in \alpha^{pos}} \mathrm{lag}_j(b) \leq (\frac{M}{M-1})^{M-2} \cdot (M-1) \cdot L_{max} \tag{25}$$

where b *is the beginning of the last priority blocking period of* X, *and* $\alpha^{pos} \equiv \{i \in \alpha(b) \,|\, \mathrm{lag}_i(b) > 0\}$.

Proof. We will proceed by induction. For the base case, let X be the first job fraction blocked by priority from the beginning of the lifetime of the system. In such a case b coincides with the beginning of the first congestion period for the MPS, which implies $\sum_{j \in \alpha^{pos}} \mathrm{lag}_j(b) \leq 0$. Hence (25) trivially holds.

For the inductive step, suppose that (25) holds for all the job fractions blocked by priority that started service before X. From the definition of last priority blocking period, it follows that the tasks in α^{pos} are less than M and cannot be

blocked by priority at time b^-. Hence, they are all under service at time b^-. From Lemma 6 and Inequality (25), we can write

$$\sum_{j\in\alpha^{pos}} \mathrm{lag}_j(b) \le |\alpha^{pos}| \cdot \tfrac{M-1}{M} \cdot L_{max} + \\ + \left[1 - (\tfrac{M-1}{M})^{|\alpha^{pos}|}\right] \cdot \left[(\tfrac{M}{M-1})^{M-2} \cdot [(M-1)\cdot L_{max}]\right]$$

Consider that, as $|\alpha^{pos}|$ increases, $(\frac{M-1}{M})^{|\alpha^{pos}|}$ decreases, hence the right term in the previous inequality increases. Hence, since $|\alpha^{pos}| \le M-1$, the thesis follows from setting $|\alpha^{pos}| = M-1$. □

4.3. Maximum lag and maximum lateness

First we prove our upper bounds to the lag.

Proof of Theorem 1 Due to space limitations and since the proof is quite intuitive, we report just a sketch here. The full proof can be found in [15].

Given a fraction X blocked by priority, it is easy to prove that $\mathrm{lag}_i(s)$ is upper bounded by the the right term of (2), by just assuming $\sigma = \emptyset$ in Lemma 3 and substituting in (16) the upper bound to the total lag provided by Theorem 3. It is easy to prove that the same bound holds if X is blocked by precedence, by considering that the lag of a task can not increase while the task is being served, and assuming that the chain head is blocked by priority. Finally, thanks to Lemma 1, the Inequality (2) holds at any time instant.

The upper bound (3) on the completion time of a job can be computed by applying the above arguments to the last fraction of the job, and subtracting, to the upper bound to $\mathrm{lag}_i(s)$, the difference between the amount of service received by the task during the service of the fraction and the maximum amount service that the task can receive in the DPS during the same time interval. □

We can now prove our upper bound to the lateness.

Proof of Theorem 2 Recall that $\mathrm{lat}_i^j = f_i^j - F_i^j$. If $f_i^j \le F_i^j$, the thesis trivially holds. Consider the case $f_i^j > F_i^j$. We will prove the thesis by contradiction. The schedules of (the fractions of) J_i^j in the MPS and in the DPS, and hence the difference $f_i^j - F_i^j$, do not depend on whether task i issues new jobs after a_i^j. Suppose that indeed an indefinite number of jobs has been issued by task i at time a_i^j. In such a case, if the upper bound (4) does not hold, we have that

$$W_i^{DPS}(F_i^j,\, f_i^j) \;>\; U_i \cdot \left[\tfrac{M-1}{M}\cdot L_{i,\,max} + (\tfrac{M}{M-1})^{M-3}\cdot L_{max}\right]$$

Furthermore, since $W_i^{DPS}(F_i^j) = W_i^{MPS}(f_i^j)$ we have

$$\begin{aligned} \mathrm{lag}_i(f_i^j) &= \\ W_i^{DPS}(f_i^j) - W_i^{MPS}(f_i^j) &= \\ W_i^{DPS}(f_i^j) - W_i^{DPS}(F_i^j) &= \\ W_i^{DPS}(F_i^j,\, f_i^j) &> \\ U_i \cdot \left[\tfrac{M-1}{M}\cdot L_i + (\tfrac{M}{M-1})^{M-3}\cdot L_{max}\right] & \end{aligned}$$

which contradicts Inequality (3). □

All is left to prove is Corollary 1.

Proof of Corollary 1 It is immediate to note that (2), (3) and (4) are non-decreasing functions of M if and only if $(\frac{M}{M-1})^{M-1}$ is a non decreasing function of M. Assuming $M \ge 1$, and defined $x \equiv M-1 \ge 0$, we have

$$(\tfrac{M}{M-1})^{M-1} = (1+\tfrac{1}{x})^x$$

Defined $f(x) \equiv (1+\frac{1}{x})^x$, we can compute the first derivative $D[f(x)] = e^{x\cdot\log\left[1+\frac{1}{x}\right]} \cdot \left[\log\left[1+\frac{1}{x}\right] - \frac{1}{x+1}\right]$. It is easy to prove that $D[f(x)] \ge 0\, \forall x \ge 0$. Hence $\forall x \ge 0 \quad f(x) \le \lim_{x->\infty}(1+\frac{1}{x})^x = e$. The thesis follows from substituting e in place of $(\frac{M}{M-1})^{M-1}$ in the right terms of (2), (6) and (4). □

5. Simulations

We simulated EDF global scheduling over 19 SMP platforms, comprised of 2, 3, ..., 20 unit-speed processors, respectively. For each SMP, we considered four different types of task sets, all with total utilization equal to the number of available processors. The first type of task sets was made only of *light tasks*, i.e. tasks with utilization no higher than 0.5. In the second type, half of the total capacity was devoted to light tasks, while the other half was devoted to *heavy tasks*, i.e. tasks with utilization higher than 0.5. The third type of task sets was made only of heavy tasks. The fourth type was made only of *very* heavy tasks, i.e. tasks with utilization higher than 0.8.

For each SMP and for each type of task set, 50 task sets were randomly generated. Finally, for each task set, the corresponding EDF schedule was simulated for $2\cdot 10^4 \cdot 10^5$ ticks, 10^5 ticks being the maximum task period.

For the first type of task sets, Fig. 3.(a) shows, for each SMP, the maximum ratio recorded, over the 50 simulation runs, between the lateness experienced by a task and the value of the upper bound (4) for the task. We will shortly refer to the above quantity as the maximum ratio. For each SMP, the mean ratio, i.e. the ratio between the mean lateness experienced by the task with the maximum ratio and the upper bound (4) is reported as well.

As can be seen, the maximum ratio decreases as the number of processors increases. Especially, it is equal to 0.73 for 2 processors, and it stabilizes at about 0.15 for a number of processors larger than 10. The mean ratio soon becomes negligible. Moving to the successive types of task sets, both the maximum and the mean ratio increase. The highest values are achieved in case of only very heavy tasks, as shown in Fig. 3.(b). Fig. 3.(c) shows the values of the upper bound (4) for the tasks that experienced the ratio reported in inset (b). Especially, the values of the upper bound reported in Fig. 3.(c) coincide with the ones we obtained for the other three types of task sets (in fact, according to (4), the bound does not depend on the utilization of the tasks).

Fig. 3.(b) shows that the bound is virtually tight for two processors. Then the maximum ratio stabilizes at approximately $1/3$ for a number of processors larger than 10.

In the end, according to the simulations, the bound is tight

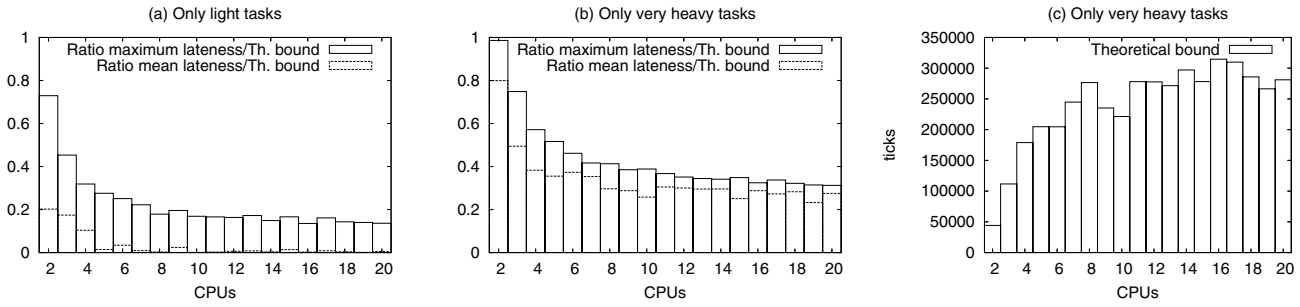

Figure 3. Inset (a) and (b): ratio between experienced lateness and upper bound. Inset (c): value of the upper bound.

only for very heavy tasks on 2 processors, while it is too conservative in the other cases. However, we could not generate all the possible light and heavy task sets during simulations. Finally, the actual bound may depend on the characteristics of the tasks (computation time and periods). Indeed, determining a possible relationship between the properties of a task set and the resulting worst-case lateness is still an open problem.

6. Conclusions

In this paper we propose an upper bound to the lateness of soft real-time tasks scheduled by EDF on a SMP. First we show that not all scheduling algorithms are able to provide a bounded lateness in the case of full utilization. Then, we propose a bound and prove its correctness. The proposed bound is in a simple closed form, and it has been shown to be virtually tight for heavy task sets on 2 processors. According to the simulations, the bound is not tight for more than 2 processors and for light task sets.

Obviously, we could not generate all the possible light and heavy task sets during simulations. Hence, proving whether the bound is tight, for light task sets, and for more than 2 processors, is still an open problem. We believe that possible relationships between the properties of the tasks and the actual bound should be investigated.

Acknowledgements

We wish to thank Enrico Bini for his insightful suggestions to improve the presentation of this paper.

References

[1] J. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.

[2] B. Andersson. *Static-priority scheduling on multiprocessors*. PhD thesis, Department of Computer Engineering, Chalmer University of Technology, Goteborg, Sweden, 2003.

[3] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In IEEE, editor, *Proceedings of the IEEE Real-Time Systems Symposium*, Dec 2001.

[4] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium, RTSS'03*, 2003.

[5] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 6, 1996.

[6] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. Chapman Hall/ CRC Press, 2004.

[7] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In IEEE, editor, *Proceedings of the IEEE Real-Time Systems Symposium*, pages 183–192, Dec 2001.

[8] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, Sep-Oct 2003.

[9] R. Graham. *Computer and Job Scheduling Theory*, chapter Bounds on the performance of scheduling algorithms. Wiley, New York, 1976.

[10] R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *14th IEEE International Conference on Distributed Computing Systems*, Los Alamitos, 1994.

[11] A. Khemka and R. K. Shyamasunda. Multiprocessor scheduling of periodic tasks in a hard real-time environment. Technical report, Tata Institute of Fundamental Research, 1990.

[12] A. Mok and M. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Proceedings of the Seventh Texas Conference on Computing Systems*, 1978.

[13] Y. Oh and S. H. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Journal on Real Time Systems*, 9, 1995.

[14] A. Srinivasan and J. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. In *Proceedings of the at the 15th Euromicro Conference on Real-time Systems*, pages 51–59, July 2003.

[15] P. Valente and G. Lipari. An upper bound to the lateness of edf on multiprocessors. Technical Report RETIS TR05-01, Scuola Superiore S. Anna, 2005. http://feanor.sssup.it/~pv/edf_tr.pdf.