Important Notice

This TR is a simplified version of our paper “An Upper Bound to the Lateness of Soft Real-time Tasks Scheduled by EDF on Multiprocessors”, published in the Proceedings of RTSS’05. Unfortunately, in that paper there is an error in the proof of Lemma 3. This TR contains only a subset of the sane proofs of that paper. The proofs used in this TR leads to a higher bound with respect to the published paper.

However, we are working on a new TR based on a set of proofs not affected by the above error. Please check at http://feanor.sssup.it/~pv or contact us for further details.

An Upper Bound to the Lateness of EDF on Multiprocessors
Technical Report

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 commonly used 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. Moreover, if tasks are allowed to dynamically enter and leave the system, global scheduling is a more appealing strategy than partitioning.

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 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.

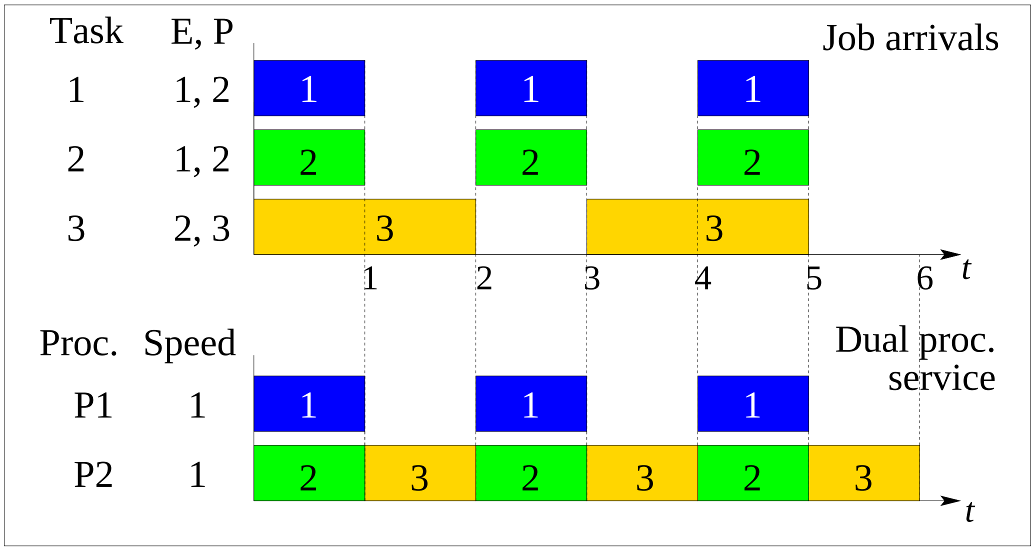

Figure 1: Example of unbounded lateness with fixed priority scheduling.

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 $4/3$.

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 Figure 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 4.1 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.

The computed upper-bound grows linearly with M. 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. According to our simulations, the grow rate of the maximum lateness experienced by the tasks is instead sub-linear with respect the number of processors. However, the bound resulted virtually tight in case of 2 processors: the ratio between the measured maximum lateness and the bound is 0.99. All the results are discussed 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, simulation results are reported 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:

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

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 $a_i^j \le t < f_i^j$ (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 $M_{\text{busy}}(t) \le M$ 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. $\forall J_i^j$ $F_i^j = d_i^j \le a_i^{j+1}$. Consequently $\forall J_i^j$ $\text{lat}_i^j = f_i^j - F_i^j$. This is the crucial property that we will exploit to compute an upper bound to the maximum lateness.

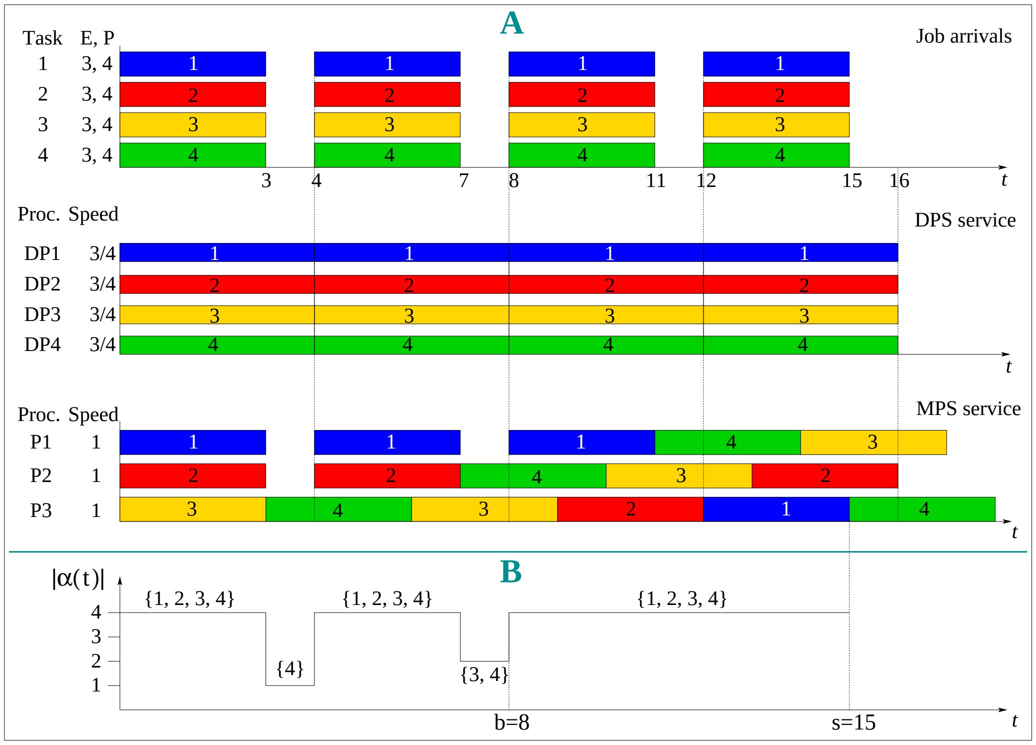

Figure 2: Comparing the MPS and the DPS.

An example of the service provided by a MPS and by its reference DPS is shown in Figure 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 Figure 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 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*

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

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.

N	Number of tasks in the task set
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 (number 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, i.e. 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
$\mathrm{lag}_i(t)$	Lag of task i ($W_i^{DPS}(t) - W_i^{MPS}(t)$).

Table 1: Notations used in this paper.

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:

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

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 $R_i^{DPS} \le R\ \forall i$, the lag of a task can not increase during the service of one of its job chains. For example, in Figure 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.

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:*

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

$$\forall J_i^j \quad \mathrm{lag}_i(f_i^j) \leq U_i \cdot \left[\frac{M-1}{M} \cdot L_i + \frac{(M-1)^2}{M} \cdot L_{max}\right]. \tag{3}$$

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:*

$$\forall J_i^j \quad \mathrm{lat}_i^j \leq \frac{M-1}{M} \cdot E_i + \frac{(M-1)^2}{M} \cdot E_{max}. \tag{4}$$

The formal proofs of the theorems 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 $\forall J_i^j \ \mathrm{lat}_i^j = 0$.

4 Proofs

In this section we will formally prove Theorems 1 and 2.

4.1 Proof notations and rationale

In this subsection we introduce the notations used in the proofs, the main idea behind them, and the proof strategy. The proofs are essentially based on computing a bound to $\mathrm{lag}_i(t)$, 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

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

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. Figure 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 Figure 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 $[b, s)$.*

Figure 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, during $[b,\, s)$ the MPS serves the jobs $\Gamma = \{J_3^2,\, J_4^2,\, J_1^3,\, J_2^3,\, J_3^3,\, J_4^3,\, J_1^4,\, J_2^4,\, J_3^4\}$.

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 4.2. This general formula will serve two purposes. We will first use it in Subsection 4.3 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.

4.2 Basic lemmas

This subsection contains three lemmas, which allow us to provide an upper bound to the lag of a task as a function of the total lag.

As shown in the last of the three lemmas, $\mathrm{lag}_i(s)$ is maximum if $F_i^j \le s$. In such a case, it is easy to understand that the value of $\mathrm{lag}_i(s)$ grows with on the amount of service $W_i^{DPS}(F_i^j,\, s)$ provided by the DPS to the *i-th* task during $[F_i^j,\, s]$. The next lemma provides an upper bound to $W_i^{DPS}(F_i^j,\, s)$, as a function of the difference $W^{MPS}(b,\, s) - W^{DPS}(b,\, F_i^j)$. This intermediate result is used in the successive lemma to find an upper bound to $W_i^{DPS}(F_i^j,\, s)$ as a function of the total lag.

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. We have:*

$$\begin{array}{rl} W_i^{DPS}(s) - W_i^{DPS}(F_i^j) & \le \\ \frac{U_i}{M} \cdot \left[W^{MPS}(b,\, s) - W^{DPS}(b,\, F_i^j) \right] & \end{array} \tag{5}$$

where b is the beginning of the last priority blocking period of X.

Proof. We define $R^{DPS}(t)$ as the total speed of the DPS at time t, $A^{DPS}(t)$ as the set of the tasks active in the DPS at time t, $B_i \equiv \{b \le t \le \min(s,\, F_i^j) \,|\, i \in A^{DPS}(t)\}$ and $\bar{B}_i \equiv [b,\, \min(s,\, F_i^j)] \backslash B_i$ (respectively, the busy and idle periods of the *i-th* task during $[b,\, \min(s,\, F_i^j)]$). First, we work on the difference $W_i^{DPS}(s) - W_i^{DPS}(\min(s,\, F_i^j))$. We can make the following algebraic manipulations:

$$\begin{array}{rl} W_i^{DPS}(s) - W_i^{DPS}(\min(s,\, F_i^j)) & = \\ (W_i^{DPS}(s) - W_i^{DPS}(b)) - (W_i^{DPS}(F_i^j) - W_i^{DPS}(b)) & = \\ W_i^{DPS}(b,\, s) - W_i^{DPS}(b,\, \min(s,\, F_i^j)) & = \\ W_i^{DPS}(b,\, s) + \int_{\bar{B}_i} \frac{U_i}{\sum_{j\in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau)\cdot d\tau - \int_{\bar{B}_i} \frac{U_i}{\sum_{j\in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau)\cdot d\tau - W_i^{DPS}(b,\, \min(s,\, F_i^j)) & \end{array} \tag{6}$$

To get the thesis we will first find an upper bound to $W_i^{DPS}(b,\, s) + \int_{\bar{B}_i} \frac{U_i}{\sum_{j\in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau)\cdot d\tau$ and then a lower bound to $\int_{\bar{B}_i} \frac{U_i}{\sum_{j\in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau)\cdot d\tau + W_i^{DPS}(b,\, \min(s,\, F_i^j))$. Since $[b,\, s]$ falls inside a full-load period (at least M tasks are active in the MPS during $[b,\, s)$), then $\forall t \in [b,\, s]$ the total speed of the MPS is $R^{MPS}(t) = M \cdot R(t)$. In contrast, we have

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

Furthermore, defined $\chi_i(t)$ as the fraction $R^{DPS}(t)$ that the DPS dedicates to the $i-th$ task at time t, we have $\forall t \in B_i\; \chi_i(t) = \frac{U_i}{\sum_{j \in A^{DPS}(t)} U_j}$. Hence:

$$\begin{array}{rl}
W_i^{DPS}(b,\, s) + \int_{\bar{B}_i} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau & \leq \\
\int_{B_i} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau + \int_{\min(s,\, F_i^j)}^{s} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau + & \\
+ \int_{\bar{B}_i} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau & \leq \\
\int_b^s \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau & = \\
\int_b^s \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot \frac{\sum_{j \in A^{DPS}(\tau)} U_j}{M} \cdot R^{MPS}(\tau) & = \\
\frac{U_i}{M} \cdot \int_b^s R^{MPS}(\tau) \cdot d\tau & = \\
\frac{U_i}{M} \cdot W^{MPS}(b,\, s) &
\end{array} \tag{7}$$

We find now a lower bound to $\int_{\bar{B}_i} \chi_i(\tau) \cdot R^{DPS}(\tau) \cdot d\tau + W_i^{DPS}(b,\, F_i^j)$. Considering that $[b,\, \min(s,\, F_i^j)] \subseteq [b,\, s]$, and that $\forall t\; \sum_{j \in A^{DPS}(t)} U_j \leq M$, we have

$$\begin{array}{rl}
\int_{\bar{B}_i} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau + W_i^{DPS}(b,\, \min(s,\, F_i^j)) & = \\
\int_{\bar{B}_i} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau + \int_{B_i} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau & \geq \\
\int_b^{\min(F_i^j,\, s)} \frac{U_i}{\sum_{j \in A^{DPS}(\tau)} U_j} \cdot R^{DPS}(\tau) \cdot d\tau & \geq \\
\int_b^{\min(F_i^j,\, s)} \frac{U_i}{M} \cdot R^{DPS}(\tau) \cdot d\tau & = \\
\frac{U_i}{M} \cdot W^{DPS}(b,\, \min(F_i^j,\, s)) &
\end{array} \tag{8}$$

Substituting (8), (7) and ((11)) in (6), we get

$$\begin{array}{rl}
W_i^{DPS}(s) - W_i^{DPS}(\min(s,\, F_i^j)) & \leq \\
\frac{U_i}{M} \cdot \left[W^{MPS}(b,\, s) - W^{DPS}(b,\, \min(F_i^j,\, s)) \right] &
\end{array} \tag{9}$$

If $F_i^j \leq s$ the thesis holds. If $s \leq F_i^j$ consider that the job J_i^j is already arrived at time s, but it is finished only at time F_i^j in the DPS. This implies that the *i-th* is continuously served in the DPS during $[s, F_i^j]$. Hence

$$\begin{array}{rl}
W_i^{DPS}(s, F_i^j) & \geq \\
\frac{U_i}{M} \cdot \left[W^{DPS}(s, F_i^j) \right] & = \\
\frac{U_i}{M} \cdot \left[W^{DPS}(b,\, F_i^j) - W^{DPS}(b,\, s)) \right] & \geq \\
\frac{U_i}{M} \cdot \left[W^{DPS}(b,\, F_i^j) - W^{MPS}(b,\, s)) \right] &
\end{array} \tag{10}$$

□

As an intermediate step for computing an upper bound to $\text{lag}_i(s)$, the next lemma provides an upper bound to the difference $W_i^{DPS}(s) - W_i^{DPS}(F_i^j)$. The bound is achieved in two steps. First it is computed 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. Then the previous lemma is applied.

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. We have:*

$$\begin{array}{rl} W_i^{DPS}(s) - W_i^{DPS}(F_i^j) & \leq \\ \frac{U_i}{M} \cdot \left[\sum_{h \in \alpha^{pos}} \text{lag}_h(b) - L^{res}(J_i^j)\right] & \end{array} \tag{11}$$

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. Le Λ be the set of the tasks that receives some service in the MPS during $[b,\, s)$, we have

$$\begin{array}{rl} W^{MPS}(b,\, s) & = \\ \sum_{l \in \Lambda} W_l^{MPS}(b,\, s) & \end{array} \tag{12}$$

The amount of service $\sum_{l \in \Lambda} W_l^{MPS}(b,\, s)$ is equal to the sum of the portion of the jobs in Γ served by the MPS during $[b,\, s)$. All the jobs in Γ have priority no lower than X. Hence the DPS must have finished both these jobs and J_i^j before finishing J_i^j at time F_i^j. Furthermore, by definition of last priority blocking period, not all the jobs in Γ have arrived at time b. This implies that $F_i^j > b$. Under this hypothesis, during $[b,\, F_i^j)$ the DPS must provide the tasks in Λ with the same total amount of service $\sum_{l \in \Lambda} W_l^{MPS}(b,\, s)$ they receive in the MPS during $[b,\, s]$, minus the extra service, with respect to the MPS, it already provided to the jobs in $\Gamma \cup J_i^j$ at time b. The latter quantity is upper bounded by $\sum_{l \in \Lambda \cup i} \text{lag}_l(b)$. In the end, we have:

$$\begin{array}{rl} W^{DPS}(b, F_j^j) & \geq \\ \sum_{l \in \Lambda} W_l^{MPS}(b,\, s) + L^{res}(J_i^j) - \sum_{l \in \Lambda \cup i} \text{lag}_l(b) & \geq \\ \sum_{l \in \Lambda} W_l^{MPS}(b,\, s) + L^{res}(J_i^j) - \sum_{l \in \alpha^{pos}} \text{lag}_l(b) & \end{array}$$

Subtracting the last inequality from (12), we get:

$$\begin{array}{rl} W^{MPS}(b,\, s) - W^{DPS}(b,\, F_i^j) & \leq \\ \sum_{l \in \Lambda} W_l^{MPS}(b,\, s) - \left(\sum_{l \in \Lambda} W_l^{MPS}(b,\, s) + L^{res}(J_i^j) - \sum_{l \in \alpha^{pos}} \text{lag}_l(b)\right) & = \\ \sum_{l \in \alpha^{pos}} \text{lag}_l(b) - L^{res}(J_i^j) & \end{array} \tag{13}$$

Substituting the previous inequality in (5), 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. This lemma will constitute the basic building block for computing an upper bound to both the total lag and the lag of any task.

Lemma 4 *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. We have:*

$$lag_i(s) \leq L^{res}(J_i^j) + \frac{U_i}{M} \cdot \left[\sum_{j \in \alpha^{pos}} lag_j(b) - L^{res}(J_i^j)\right] \tag{14}$$

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 $\alpha^{pos} \equiv \{i \in \alpha(b) \,|\, lag_i(b) > 0\}$.

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 $\text{lag}_i(F_i^j,\, s)$, finally we will sum this bound to $\text{lag}_i(F_i^j)$. We have that:

$$W_i^{DPS}(F_i^j) = W_i^{MPS}(s) + L^{res}(J_i^j) \tag{15}$$

We can do some algebraic manipulations:

$$\begin{aligned} W_i^{MPS}(s) &= \\ W_i^{MPS}(F_i^j) + \left[W_i^{MPS}(s) - W_i^{MPS}(F_i^j)\right] &= \\ W_i^{MPS}(F_i^j) + \Delta & \end{aligned} \tag{16}$$

where $\Delta \equiv W_i^{MPS}(s) - W_i^{MPS}(F_i^j)$. Substituting successively (15) and (16) into the definition of $\text{lag}_i(F_i^j)$, we get

$$\begin{aligned} \text{lag}_i(F_i^j) &= \\ W_i^{DPS}(F_i^j) - W_i^{MPS}(F_i^j) &= \\ W_i^{MPS}(s) + L^{res}(J_i^j) - W_i^{MPS}(F_i^j) &= \\ L^{res}(J_i^j) + \left[W_i^{MPS}(s) - W_i^{MPS}(F_i^j)\right] &= \\ L^{res}(J_i^j) + \Delta & \end{aligned} \tag{17}$$

Furthermore:

$$\begin{aligned} \text{lag}_i(F_i^j,\, s) &= \\ \left[W_i^{DPS}(s) - W_i^{MPS}(s)\right] - \left[W_i^{DPS}(F_i^j) - W_i^{MPS}(F_i^j)\right] &= \\ \left[W_i^{DPS}(s) - W_i^{DPS}(F_i^j)\right] - \left[W_i^{MPS}(s) - W_i^{MPS}(F_i^j)\right] &= \\ \left[W_i^{DPS}(s) - W_i^{DPS}(F_i^j)\right] - \Delta & \end{aligned} \tag{18}$$

where the last identity follows from (16). Thanks Lemma 3, we have

$$\begin{aligned} W_i^{DPS}(F_i^j,\, s) &\leq \\ \frac{U_i}{M} \cdot \left[\sum_{j \in \alpha^{pos}} \text{lag}_j(b) - L^{res}(J_i^j)\right] & \end{aligned} \tag{19}$$

Substituting the last inequality in (18), we get

$$\begin{aligned} \text{lag}_i(F_i^j,\, s) &= \\ (W_i^{DPS}(s) - W_i^{DPS}(F_i^j)) - \Delta &\leq \\ \frac{U_i}{M} \cdot \left[\sum_{j \in \alpha^{pos}} \text{lag}_j(b) - L^{res}(J_i^j)\right] - \Delta & \end{aligned} \tag{20}$$

Finally, summing this inequality to (17), we get the thesis.

□

4.3 Bounding the total lag

We need a last intermediate lemma.

Lemma 5 *Let $A(\bar{t})$ be any subset, comprised of no more than $M - 1$ tasks, of the set of the tasks under service at time $\bar{t}$. We have that*

$$\sum_{j \in A(\bar{t})} lag_j(\bar{t}) \leq (M-1)^2 \cdot L_{max} \tag{21}$$

Proof. We will proceed by induction.

The time interval $[0,\, \bar{t}]$ can be divided into a finite sequence of sub-intervals, such that during each sub-interval, the set of the tasks under service does not change. In the rest of this proof, we will call just a *sub-interval* each of the above defined sub-intervals. Consequently, during each sub-interval the sum of the lags of the tasks under service cannot increase. Hence the sum of the lag of any subset of the tasks under service during any sub-interval is upper bounded by the sum of the lags of the same tasks at the beginning of the sub-interval. Let $\bar{t}'$ be the beginning of the sub-interval $\bar{t}$ belongs to, we have that $A(\bar{t}) = A(\bar{t}')$ and that

$$\sum_{j \in A(\bar{t})} \text{lag}_j(\bar{t}) \le \sum_{j \in A(\bar{t})} \text{lag}_j(\bar{t}') \tag{22}$$

Hence, in the rest of this proof, we prove that the thesis holds at the beginning $\bar{t}'$ of the sub-interval $\bar{t}$ belongs to. For the base case, the thesis trivially holds if $\bar{t}' = 0$. As inductive hypothesis, suppose that the thesis holds for any subset, comprised of no more than $M-1$ tasks, of the tasks under service at the beginning of any of the sub-intervals preceding the one $\bar{t}$ belongs to.

We can assume, without losing generality, that the $V < M$ 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. Let b_j be the last priority blocking period of the fraction X_j, let $\alpha_j(t)$ be the set of the tasks that have pending jobs with priority no lower than X_j at time t, and let $\alpha_j^{pos} \equiv \{i \in \alpha_j(b_j) \,|\, \text{lag}_i(b_j) > 0\}$. From Lemma 4 we can write

$$\begin{array}{rl} \sum_{j \in A(\bar{t})} \text{lag}_j(\bar{t}') & \le \\ \sum_{j \in A(\bar{t})} \text{lag}_j(s_j) & \le \\ \sum_{j \in A(\bar{t})} \left\{ L^{res}(J(X_j)) + \frac{U_j}{M} \cdot \left[\sum_{j \in \alpha_j^{pos}} \text{lag}_j(b_j) - L^{res}(J(X_j)) \right] \right\} & \end{array} \tag{23}$$

because the lag of a task can not increase during the execution of one of its chains, and the task $j \in [1, V]$ is continuously served during $[s_j, \bar{t}]$. From the inductive hypothesis and the fact that $\forall j \in A(\bar{t}) \ \left|\alpha_j^{pos}(b_j)\right| \le M-1$, we have

$$\max_{j \in A(\bar{t})} \left[\sum_{i \in \alpha_j^{pos}} \text{lag}_i(b_j) \right] \le (M-1)^2 \cdot L_{max}$$

Hence

$$\begin{array}{rl} \sum_{j \in A(\bar{t})} \text{lag}_j(\bar{t}') & \le \\ \sum_{j \in A(\bar{t})} \left\{ L^{res}(J(X_j)) + \frac{U_j}{M} \cdot \left[(M-1)^2 \cdot L_{max} - L^{res}(J(X_j)) \right] \right\} & = \\ \sum_{j \in A(\bar{t})} \left\{ \frac{M-U_j}{M} \cdot L^{res}(J(X_j)) + \frac{U_j}{M} \cdot (M-1)^2 \cdot L_{max} \right\} & \le \\ \sum_{j \in A(\bar{t})} \left\{ \frac{M-U_j}{M} \cdot L_{max} + \frac{U_j}{M} \cdot (M-1)^2 \cdot L_{max} \right\} & = \\ L_{max} \cdot \sum_{j \in A(\bar{t})} \left\{ 1 - \frac{U_j}{M} + \frac{U_j}{M} \cdot (M^2 - 2 \cdot M + 1) \right\} & = \\ L_{max} \cdot \sum_{j \in A(\bar{t})} \left\{ 1 - \frac{U_j}{M} + \left(U_j \cdot M - 2 \cdot U_j + \frac{U_j}{M} \right) \right\} & = \\ L_{max} \cdot \sum_{j \in A(\bar{t})} \left\{ 1 - \frac{U_j}{M} + U_j \cdot M - 2 \cdot U_j + \frac{U_j}{M} \right\} & = \\ L_{max} \cdot \sum_{j \in A(\bar{t})} \{ 1 + U_j \cdot M - 2 \cdot U_j \} & = \\ L_{max} \cdot \sum_{j \in A(\bar{t})} \{ 1 + U_j \, (M-2) \} & \le \\ L_{max} \cdot \sum_{j \in A(\bar{t})} \{ 1 + (M-2) \} & = \\ L_{max} \cdot \sum_{j \in A(\bar{t})} (M-1) & = \\ V \cdot (M-1) \cdot L_{max} & \le \\ (M-1)^2 \cdot L_{max} & \end{array} \tag{24}$$

□

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

Theorem 3 *Let X be a generic job fraction, belonging to a job J_i^j, which starts service in the MPS after being blocked by priority, that starts service at time s in the MPS after being blocked by priority. We have*

$$\sum_{q \in \alpha^{pos}} lag_q(b) \leq (M-1)^2 \cdot L_{max} \tag{25}$$

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

Proof. The thesis trivially follows from the previous lemma. □

4.4 Maximum lag and maximum lateness

We can now prove Theorems 1 and 2.

Proof of Theorem 1 Thanks to Lemma 1, to prove (2), all we need is to compute an upper bound to the lag experienced by task i at the start time s of any job fraction X belonging to a job J_i^j. Apart from the trivial case when the job J_i^j starts service as it arrives, and is served until completion, we will distinguish between two cases:

1. X has been blocked by priority. Thanks to Lemma 4, we have

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

   where b is the beginning of the last priority blocking period for X. Substituting (25) (Theorem 3) into the last expression, we get

   $$\begin{array}{rl} \text{lag}_i(s) & \leq \\ L^{res}(J_i^j) + \frac{U_i}{M} \cdot \left[(M-1)^2 \cdot L_{max} - L^{res}(J_i^j)\right] & = \\ L^{res}(J_i^j) + U_i \cdot \left[\frac{(M-1)^2}{M} \cdot L_{max} - \frac{L^{res}(J_i^j)}{M}\right] & = \\ (1 - \frac{U_i}{M}) \cdot L^{res}(J_i^j) + U_i \cdot \frac{(M-1)^2}{M} \cdot L_{max} & \end{array} \tag{26}$$

   which proves (2).

   To prove (3), assume that X is the last fraction of J_i^j. We have $W_i^{MPS}(s,\ f_i^j) = L^{res}(J_i^j)$, while during the same time interval, $W_i^{DPS}(s,\ f_i^j) \leq U_i \cdot L^{res}(J_i^j)$ (recall that $\forall t\ R_i^{DPS}(t) = U_i \cdot R(t)$). As a consequence:

   $$\begin{array}{rl} \text{lag}_i(f_i^j) - \text{lag}_i(s) & = \\ W_i^{DPS}(s,\ f_i^j) - W_i^{MPS}(s,\ f_i^j) & \leq \\ (U_i - 1) \cdot L^{res}(J_i^j) & \end{array}$$

   As a conclusion, considering also (26):

   $$\begin{array}{rl} \text{lag}_i(f_i^j) & = \\ \text{lag}_i(s) + \text{lag}_i(f_i^j) - \text{lag}_i(s) & \leq \\ (1 - \frac{U_i}{M}) \cdot L^{res}(J_i^j) + (U_i - 1) \cdot L^{res}(J_i^j) + U_i \cdot \frac{(M-1)^2}{M} \cdot L_{max} & = \\ (1 - \frac{1}{M}) \cdot U_i \cdot L^{res}(J_i^j) + U_i \cdot \frac{(M-1)^2}{M} \cdot L_{max} & \end{array}$$

   which proves (3).

2. X has been blocked by precedence. Let X^{first} be the head of the chain X belongs to. Considering that X^{first} necessarily falls into the previous case, that $\text{lag}_i(s) \leq \text{lag}_i(s_{first})$, and Inequality (26), Inequality (2) follows. Using the same arguments as in the previous case, (3) can be proven as well. □

We can finally prove our upper bound on the maximum lateness.

Proof of Theorem 2 Recall that $\text{lat}_i^j = f_i^j - F_i^j$. If $f_i^j \le F_i^j$, the thesis trivially holds. Consider the case $f_i^j > F_i^j$. 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. We will prove the thesis by contradiction. Suppose that:

$$\begin{array}{rcl} f_i^j - F_i^j & > \\ \frac{1}{R} \cdot \left[\frac{M-1}{M} \cdot L_i + \frac{(M-1)^2}{M} \cdot L_{max}\right] & = \\ \frac{1}{U_i \cdot R} \cdot U_i \cdot \left[\frac{M-1}{M} \cdot L_i + \frac{(M-1)^2}{M} \cdot L_{max}\right] \end{array}$$

In such a case we have that:

$$W_i^{DPS}(F_i^j, f_i^j) > U_i \cdot \left[\frac{M-1}{M} \cdot L_{i,\,max} + \frac{(M-1)^2}{M} \cdot L_{max}\right]$$

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

$$\begin{array}{rcl} \text{lag}_i(f_i^j) & = \\ W_i^{DPS}(f_i^j) - W_i^{MPS}(f_i^j) & = \\ W_i^{DPS}(f_i^j) - W_i^{DPS}(F_i^j) & = \\ W_i^{DPS}(F_i^j, f_i^j) & > \\ U_i \cdot \left[\frac{M-1}{M} \cdot L_i + \frac{(M-1)^2}{M} \cdot L_{max}\right] \end{array}$$

which contradicts Inequality (3). □

5 Simulations

We are simulating EDF global scheduling over 9 SMP platforms, comprised of $2, 3, \ldots, 10$ unit-speed processors, respectively. For each SMP, we considered *very* heavy task sets, i.e. task sets made of tasks with utilization higher than 0.8 each.

For each SMP 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.

According to the simulations, the bound is tight only for very heavy tasks on 2 processors, while it is too conservative in the other cases. Especially, the distance between the bound and the observed lateness grows with the number of processors.

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.

Especially, the distance between the bound and the observed lateness grows with the number of processors.

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.