Bounding Execution Resources for the Task Scheduling Problem in Cyber-Physical Systems

Vlad Rădulescu
A. I. Cuza University of Iaşi
Department of Computer Science
Iaşi, Romania
rvlad@info.uaic.ro

Ştefan Andrei
Lamar University
Department of Computer Science
Beaumont, TX, USA
sandrei@cs.lamar.edu

Albert M.K. Cheng
University of Houston
Department of Computer Science
Houston, TX, USA
cheng@cs.uh.edu

ABSTRACT

In designing and analyzing real-time systems, the central problem resides in finding a feasible schedule for a given task set, if one exists. A lot of research effort has been carried out in approaching the various aspects of task scheduling. While most results have been achieved for preemptive scheduling, the non-preemptive case has still room for improvement, due to its complexity. In addition, the widespread usage of cyber-physical systems (CPS) is putting real-time scheduling in the position to deal with new challenges, as additional (and sometimes particular) requirements are raised by such systems.

This paper, which continues the previous work of the authors, introduces a new lower bound on the number of processing units that allows a feasible schedule of a task set for both preemptive and non-preemptive scheduling. As a specific CPS issue, the necessity of moving the processing units in space, which incurs additional time requirements, is considered. The single-instance case is first discussed, then the results are extended to the periodic case.

KEYWORDS

multiprocessor scheduling; lower and upper bounds for the number of processors

1 INTRODUCTION

Recently, it has become clear that the diversity of potential applications for cyber-physical systems is tremendous. Unfortunately, so has the variety of requirements brought by these systems. If we are to look only at some of the fields that already benefit from the use of CPSs, such as robotics, automotive industry, healthcare monitoring, process control, or flight control, we easily see that each of these fields comes with its own specific problems to be addressed, entirely different from the others. As most cyber-physical systems need to have real-time characteristics, this diversity has come to question the effectiveness of the models and methodologies developed for “classic” real-time systems [21].

Until new paradigms will emerge, one way for the real-time systems to adapt to the CPS world might be to return to simpler structures and approaches. This approach will eliminate some of the features that are helpful for other applications, but here can affect the desired behavior. One such option is to pay more attention to non-preemptive scheduling.

Preemptive scheduling has been largely studied, and numerous results have been achieved for a quite large range of problems. Two of the best-known algorithms, namely EDF (Earliest Deadline First) [14] and LLF (Least Laxity First) [25], are optimal for the case of single-processor, preemptive scheduling. These two algorithms are still the starting points for many recent techniques, which attempt to combine and improve them.

EDZL (Earliest Deadline first until Zero Laxity) [22],[10] handles the particular case of zero-laxity tasks by assigning them the highest priority, while for the rest of the time it only considers deadlines, in an EDF-based manner.

FPZL (Fixed Priority until Zero Laxity) [12] and its two variants, FPSL (Fixed Priority until Static Laxity) and FPCL (Fixed Priority until Critical Laxity) [13] use global fixed priority preemptive scheduling, except again for the case of zero-laxity tasks, which receive the highest priority, just as for the previous technique. Despite all attempts of improvement, however, for most subclasses of the scheduling problem, there is no optimal algorithm [30].

However, preemptive scheduling, with all its advantages, also has some performance issues: the runtime overhead introduced by preemption itself, which is not only significant, but sometimes even unpredictable, depending on the scheduling algorithm [7]; the complexity of mutual exclusion [29]; higher memory consumption [24]; the effects of preemption on program locality (cache misses, wrong working of prefetch mechanisms), which also affects the predictability of the system's behavior [27]. Worse, in some cases preemption may be impossible or too expensive to be used in practice [17]. While many of these factors are rather general,

ESWeek Workshop on Declarative Embedded and Cyber-Physical Systems, October 19, 2017, Seoul, South Korea. Copyright held by the owner/author(s).

the insufficient predictability caused by some of them is particularly undesirable in the case of cyber-physical systems [21]. Thus, the door opens for non-preemptive scheduling, which re-emerges as an option.

The price to be paid consists in the higher difficulty of non-preemptive scheduling. In fact, the non-preemptive scheduling problem has been proven to be $\mathcal{NP}$-hard [8], [16], [9], [5], [18], [19], so it comes as no surprise that the majority of the results have been achieved for preemptive scheduling.

It is hard to get a general description of the challenges the cyber-physical systems are facing, due to the very nature of the problems they have to solve. Nevertheless, some issues show up in many cases, like network interconnection and low consumption. It is c that the most common of them is the mobile nature of the devices which have to be controlled.

This paper is not focusing on the scheduling itself, but on the number of processing units that are necessary in the system. Usually, starting from a given number of processors in a system, one attempts to determine whether a feasible schedule can be found. Such an approach is justified by the fact that the preemptive scheduling problem can be solved through linear programming if the number of processors is fixed [1]; however, if the number of processors is unbounded, the scheduling problem becomes $\mathcal{NP}$-hard [20]. On the other hand, the non-preemptive scheduling problem is $\mathcal{NP}$-hard in all cases [6], [19]. That is why results that hold both for preemptive and non-preemptive scheduling are needed.

In this paper, a different approach is taken, starting from a given task set and determining the minimum number of processing units that might allow the existence of a feasible schedule; the time delay incurred by the movement of the processing unit is also taken into account. When non-preemptive scheduling is considered, this minimum number does not guarantee schedulability; instead, it sets up a lower bound on the number of processors. In the case of preemptive scheduling, the lower bound is closer to providing guarantees of schedulability, although a 100% certainty still remains to be proved. An upper bound is also determined, in order to set up a limit beyond which any addition of new processing units to the system will be useless.

The cases of single-instance tasks and periodic tasks, respectively, are considered in the paper, for both the lower and the upper bound.

The remainder of the paper is organized as follows. Section 2 shows the notations used in the paper and the assumptions under which the lower and upper are computed. Section 3 introduces the new lower bound on the number of processing units for single-instance tasks, which an improved version of previous work of the authors and, besides, is also adapted to the requirements of cyber-physical systems. In Section 4, the result is extended to the case of periodic tasks. Section 5 briefly discusses the upper bound. Finally, Section 6 draws the conclusions and lays the path for future research.

2 THE MODEL

A single-instance task is executed exactly once. Thus, for a set of single-instance $\mathcal{T} = \{T_1, \ldots, T_n\}$, each task T_i is described as $T_i = (s_i, c_i, d_i)$, where:
s_i – the start time of the task
c_i – the computational cost of the task (i.e., the execution time)
d_i – the deadline of the task

In the case of cyber-physical systems, additional elements must be considered. One of the many facets of handling CPSs is the necessity of moving the servicing unit (i.e., the processing unit) to the location where the service is performed and back to the base location of the servicing unit. Apart from the additional power consumption, there is always an additional time required for reaching the destination point and then returning to the base location [26]. This, of course, must be explicitly considered by the scheduling algorithm. For simplicity, for each task T_i, the delays introduced by movements to and from the servicing location are considered equal, so they are described by the same delay value, denoted by m_i.

Another issue is that a CPS is typically a highly heterogeneous system, whose components have various specifications and yield various performance capabilities [23]. For example, the different response times of the different sensors have an impact on the times when the tasks may be scheduled. This paper is considering sets of independent tasks, thus the start times of the tasks are determined by the requirements on the actions the perform and not by inter-task dependencies. In our model, then, for each task T_i, the delays introduced by the response times of the sensors used by T_i are included in the start time s_i.

A periodic task is executed repeatedly, on a periodic basis. In this case, each task T_i is defined as $T_i = (S_i, c_i, D_i, p_i)$, where:
S_i - the initial start time
c_i - the computational cost
D_i - the initial deadline
p_i - the task period

Based on the definition above, the following notations are used to describe a particular instance of a task T_i:
$s_{i,k} = S_i + k \cdot p_i$ - the start time of instance k
$c_{i,k} = c_i$ - the computational cost of instance k
$d_{i,k} = D_i + k \cdot p_i$ - the deadline of instance k

Additionally, we consider that $s_{i,k+1} > d_{i,k}, \forall k \in \mathbb{N}$, that is, a certain instance of a task may not start before the deadline of its previous instance.

Of course, the modeling of cyber-physical systems is the same as described above for single-instance tasks.

3 THE LOWER BOUND

In [3], the authors proposed a formula for computing the lower bound on the required number of processors for a multi-processor, non-preemptive, independent task set $\mathcal{T} = \{T_1, \ldots, T_n\}$. While the work was focusing on non-preemptive scheduling, the result is also useful in the preemptive case. The idea is to put together, for each time interval (t_1, t_2), the minimum CPU time required until the end of the interval, which is t_2 (that is, the processor time necessary such that all tasks may meet their deadlines), and the maximum CPU time available for execution before the start of the interval, namely t_1 (corresponding to the tasks whose start times have been reached, so they may be scheduled for execution). The difference between these two times gives the minimum amount of CPU time that is necessary for execution during the interval (t_1, t_2), which, in turn, leads to the minimum number of processors necessary during that interval. Since the total number of processors must cover the worst-case time interval, it has to be computed as the maximum value over all possible time intervals. Having a number of processors equal to the lower bound does not guarantee that a feasible schedule can be found, but it is certain that, on a system with fewer processors, a feasible schedule is not possible.

Obviously, this technique requires knowing in advance the start times, deadlines, and computational times of the tasks, which is a limitation for some real-time systems. Nevertheless, in practice there are many systems that comply with this restriction; besides, any kind of on-the-fly estimation of the number of processors, without prior knowledge on the tasks, is clearly impossible.

In attempting to find such a lower bound, we start by observing that the minimum CPU time required changes only when at least one task's deadline is reached; similarly, the maximum CPU time available changes only when at least one task's start time is reached. Thus, we do not need to consider all possible time intervals. Instead, the computations must be performed only for all intervals (s_j, d_i), where s_j is the start time of task T_j, d_i is the deadline of task T_i, and $d_i > s_j$.

So, for each task T_i in the set, we have the following terms:

- The requested execution time (R_i), the minimum amount of CPU time that must be available to the tasks before the deadline of T_i.
- The available execution time (A_i), the maximum amount of CPU time that may be available to the tasks before the start time of T_i.

For any time interval (s_j, d_i), by subtracting the available execution time from the requested execution time, we get the minimum amount of execution time that is needed within that time interval; furthermore, dividing this value against the length of the interval (i.e., $d_i - s_j$) results in the minimum number of processors that are necessary during the interval. Of course, the lower bound is obtained by performing this computation for all time intervals (s_j, d_i).

Table 1: Example - requested and available execution times.

Task index (i)	s_i	c_i	d_i	R_i	A_i^1	A_i^2	A_i^3
1	0	3	9	10	0	1	5
2	1	4	7	8	0	1	3
3	3	3	6	6	0	0	2

The requested execution time R_i is computed by summing up the elements below:

- For each task T_j with $d_j \leq d_i$, the whole computational cost c_j must be considered; otherwise, task T_j would not meet its deadline.
- For each task T_j with $d_i < d_j < d_i + c_j$, an execution time of $c_j - (d_j - d_i)$ is considered; if task T_j executed less time before d_i, it would no longer be possible for it to meet its deadline.

As an example, let us consider the task set in Table 1. We see that:

- $R_1 = 3 + 4 + 3 = 10$, because all tasks must be terminated by time d_1, which is the latest deadline
- $R_2 = 1 + 4 + 3 = 8$, as tasks T_2 and T_3 must be terminated by deadline d_2, while task T_1 must execute at least 1 time unit until the same moment
- $R_3 = 0 + 3 + 3 = 6$: by deadline d_3, task T_3 must be terminated, task T_2 must execute at least 3 time units, and task T_1 may start executing afterwards

For cyber-physical systems, as seen before, the delay required for moving the processing unit to the destination location and back to the base location must also be handled. In the expression of R_i, these delays (both of the same value m_i in our model) must be added to the computational cost, as together they give the amount of time during which a processing unit is used exclusively for servicing a certain location (i.e., where the corresponding task is executed). In addition, the deadline must be extended by m_i; the rationale is that, if the execution of the task must be completed until time d_i, then the processing unit must be released (i.e., become available again) until time $d_i + m_i$, when returning to the base location must also be done.

For the sake of simplicity, we denote the adjusted computational cost of a task T_i, which also considers the transportation delays, by $c_i^{'} = c_i + m_h \cdot 2$. Similarly, the adjusted deadline is denoted by $d_i^{'} = d_i + m_i$.

The analytical expression of the sum above, based on [3] and adding the delays discussed above, can then be expressed as:

$$R_i = \sum_{\substack{h \\ d'_h \le d'_i}} c'_h + \sum_{\substack{h \\ d'_i < d'_h < d'_i + c'_h}} (c'_h - (d'_h - d'_i))$$

In a similar manner, the available execution time A_i, as proposed in [3], consists in the following components:

- For each task T_j with $s_j + c_j \le s_i$, the entire computational cost c_j is considered, because there is enough time for task T_j to complete its execution by s_i.
- For each task T_j with $s_j \le s_i < s_j + c_j$, an execution time of $(s_i - s_j)$ is considered, as that is the maximum time that task T_j could execute until s_i.

This definition of the available execution time can be further improved. The available execution time considers the tasks that can be executed before start time s_j, provided there are enough processors in the system. We say that a task can be executed until a certain moment t if its own start time is lower that t. However, it must be noted that not all the tasks that can be executed before s_j really need to be executed so soon; some of them could be delayed for a significant amount of time and still meet their deadlines. Considering these tasks in the expression of the available execution time could be misleading, as they do not contribute in any way to meeting the deadlines of the tasks with lower laxities, which must be executed in the near future. Furthermore, even for the tasks that need to be executed at least a certain amount of time in the near future, executing them for more than that minimum amount of time required would not be helpful, as they might block the scheduling of other tasks that need to be executed. Moreover, that would lead to tasks with lower laxities being left behind instead of being scheduled. The purpose of the lower bound is to find only the mandatory execution time within each time interval (s_j, d_i), that is, the time spans that tasks must be executed before time d_i in order to meet their deadlines; thus, such auxiliary execution slices must be excluded.

Thus, the available execution time can be redesigned with respect to [3], by considering only tasks that can be executed before a certain start time s_j and, at the same time, must be executed, at least partly, before a certain deadline d_i is reached. This requires the available execution time to be defined with respect not only to the start time s_j, but also to the deadline d_i.

For each deadline d_i and each start time s_b, with $s_b < d_i$, we define the available execution time, denoted by A_i^b, as the amount of processor time available, until time s_b, for the mandatory execution of all tasks that need complete or partial execution before time d_i.

As is easy to see, the set of tasks that are considered in the expression of A_i^b is a subset of the tasks considered in the expression of R_i. This subset can further be split into four categories:

- Tasks T_h with $s_h + c_h \le s_b$ and $d_h \le d_i$, for which the whole computational cost c_h is considered.
- Tasks T_h with $s_h < s_b < s_h + c_h$ and $d_h \le d_i$, for which an execution time of $s_b - s_h$ is considered.
- Tasks T_h with $s_h + c_h \le s_b$ and $d_i < d_h < d_i + c_h$, for which an execution time of $c_h - (d_h - d_i)$ is considered; this is because they do not need to be executed more than $c_h - (d_h - d_i)$ time before d_i, although they could be terminated before s_b.
- Tasks T_h with $s_h < s_b < s_h + c_h$ and $d_i < d_h < d_i + c_h$, for which the minimum between $s_b - s_h$ and $c_h - (d_h - d_i)$ is considered; that is, the minimum between what could be executed before s_b and what needs to be executed before d_i.

Put into words, we look for tasks that must be executed a certain amount of time (either completely or partially) before deadline d_i, in order to be able to meet their own deadlines. Consider we find such a task T_h, which must be executed at least a time t before deadline d_i, otherwise it will miss its deadline. Then we try to execute T_h for as much time as possible before start time s_b, but not more than t, which would be useless; of course, this also depends on the start time s_h and its relation to s_b. The categories above cover all possible cases for such tasks and how much of them can be executed. By considering all pairs (s_b, d_i), we get a general picture of what tasks can be executed and when.

Let us look again at Table 1. It is obvious that $A_i^1 = 0$, $\forall i \in \{1, 2, 3\}$, as s_1 is the earliest start time, so nothing can be executed before that. Furthermore, $A_3^2 = 0$, because tasks T_2 and T_3 cannot execute before s_2, due to their start times, while task T_1 does not need to be executed at all before s_2, so it is not considered here. At the same time, $A_1^2 = A_2^2 = 0$; again, tasks T_2 and T_3 cannot execute before s_2, and task T_1, which needs to start before d_2 (or d_1), can only be executed 1 time unit before s_2.

Finally, $A_1^3 = 5$, because task T_1 can execute 3 time units before s_3 and T_2 can execute 2 time units, with respect to deadline d_3; $A_2^3 = 3$, because T_1 needs to execute just 1 time unit until d_2 in order to meet its deadline, while T_2 can execute 2 time units before s_3; and $A_3^3 = 2$, as only T_2 needs to execute before d_3 in order to meet its deadline, and it can execute only 2 time units before s_3. In all these cases, task T_3 cannot start until s_3, so it does not contribute to A_i^3.

In a cyber-physical system, since the start times of the tasks are known in advance, it is possible for a processing unit to be sent to the servicing location before its start time. As a result, instead of considering the start time s_i, the servicing may start at moment $s_i - m_i$, including here the movement towards the servicing location. For simplicity, we consider here that $m_i < s_i$, $\forall i$, that is, there is enough time to send a processing unit to any servicing location before its start time is reached. If there are exceptions in practice, then some processing units can be placed in the corresponding locations before the whole process begins. Hence, beside the notations introduced with the requested execution times, we will use an additional one: the adjusted start time is denoted by $s_i^{'} = s_i - m_i$.

In principle, it is possible to send the processing unit to the location where task t_i will be executed even before time $s_i - m_i$; however, this would bring no improvement, as the processing unit would simply have to wait there until time s_i. This is in turn it could make scheduling more difficult, as one processing unit would be allocated to a certain task for a longer time than necessary.

With all these considerations, included those stated for the requested execution times, the formula for A_i^b becomes the one below:

$$A_i^b = \sum_{\substack{h \\ s_h^{'}+c_h^{'} \leq s_b^{'} \\ d_h^{'} \leq d_i^{'}}} c_h^{'} + \sum_{\substack{h \\ s_h^{'} < s_b^{'} < s_h^{'}+c_h^{'} \\ d_h^{'} \leq d_i^{'}}} (s_b^{'} - s_h^{'}) +$$

$$+ \sum_{\substack{h \\ s_h^{'}+c_h^{'} \leq s_b^{'} \\ d_i^{'} < d_h^{'} < d_i^{'}+c_h^{'}}} (c_h^{'} - (d_h^{'} - d_i^{'}))$$

$$+ \sum_{\substack{h \\ s_h^{'} < s_b^{'} < s_h^{'}+c_h^{'} \\ d_i^{'} < d_h^{'} < d_i^{'}+c_h^{'}}} \min(s_b^{'} - s_h^{'}, c_h^{'} - (d_h^{'} - d_i^{'}))$$

We can now introduce the new measure for the lower bound:

Definition 3.1. *Given a single-instance, independent task set $\mathcal{T} = \{T_1, \ldots, T_n\}$, we define the Improved Utilization of Single-Instance task set $\mathcal{T}$ until deadline d_i as $IUSI_i = \max_{d_i > s_j} \lceil (R_i - A_i^j)/(d_i^{'} - s_j^{'}) \rceil$. We also define the Improved Utilization of Single-Instance task set $\mathcal{T}$ as $IUSI = \max_i(IUSI_i)$.* ∎

Theorem 3.1. *Given a multi-processor, independent task set $\mathcal{T} = \{T_1, \ldots, T_n\}$, there is no feasible schedule on a system with less than IUSI processors.*

Proof Let us consider a deadline $d_i^{'}$ and a start time $s_j^{'}$, with $s_j^{'} < d_i^{'}$. The expression of R_i includes all the tasks that need to be executed, fully or partly, before $d_i^{'}$. If a task with the deadline less than or equal to $d_i^{'}$ did not finish until $d_i^{'}$, its deadline will be missed and the whole schedule is unfeasible. Also, if a task that needs to be executed partially before $d_i^{'}$ did not execute enough time until $d_i^{'}$ (that is, at least the part indicated in the expression of R_i), then it will be impossible for it to meet its deadline, even if it will be scheduled continuously after $d_i^{'}$; in other words, at time $d_i^{'}$, its laxity will already be negative. All in all, R_i shows the least times that these task must be executed before $d_i^{'}$.

On the other hand, A_i^j refers to precisely the same tasks and shows how much time they can all execute before $s_j^{'}$, but no more than is needed to execute before $d_i^{'}$. This measure is based only on the start times of the tasks and gives a best-case execution variant, provided the system contains enough processors.

Putting them together, $R_i - A_i^j$ gives the minimum CPU time that is necessary for these tasks precisely in the interval $(s_j^{'}, d_i^{'})$ - no sooner, no later. In order to be able to use $R_i - A_i^j$ CPU time in an interval of length $d_i^{'} - s_j^{'}$, at least $\lceil (R_i - A_i^j)/(d_i^{'} - s_j^{'}) \rceil$ processors must be in the system. Since this condition must be satisfied for all intervals $(s_j^{'}, d_i^{'})$, the conclusion is that no feasible schedule can be found with less that *IUSI* processors. ∎

Let us now consider a more complex example than the one from Table 1:

Example 3.1. *Let us consider a single-instance, independent task set $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$, where the tasks are given by: $T_1 = (5, 6, 11)$, $T_2 = (11, 5, 16)$, $T_3 = (4, 4, 14)$, $T_4 = (8, 14, 18)$, $T_5 = (13, 3, 20)$, and the movement times are: $m_1 = 1$, $m_2 = 2$, $m_3 = 2$, $m_4 = 1$, $m_5 = 1$. By performing the computation, we get:*
$R_1 = 15, R_2 = 32, R_3 = 26, R_4 = 34, R_5 = 36$
$A_1^1 = 2, A_1^2 = 9, A_1^3 = 0, A_1^4 = 7, A_1^5 = 15$
$A_2^1 = 2, A_2^2 = 14, A_2^3 = 0, A_2^4 = 8, A_2^5 = 24$
$A_3^1 = 2, A_3^2 = 14, A_3^3 = 0, A_3^4 = 8, A_3^5 = 22$
$A_4^1 = 2, A_4^2 = 14, A_4^3 = 0, A_4^4 = 8, A_4^5 = 24$
$A_5^1 = 2, A_5^2 = 14, A_5^3 = 0, A_5^4 = 8, A_5^5 = 24$
$IUSI_1 = \max(\lceil (15-2)/(12-4) \rceil, \lceil (15-9)/(12-9) \rceil, \lceil (15-0)/(12-2) \rceil, \lceil (15-7)/(12-7) \rceil, \lceil (15-24)/(12-12) \rceil) = \max(2, 2, 2, 2, 0) = 2$
$IUSI_2 = \max(\lceil (32-2)/(18-4) \rceil, \lceil (32-14)/(18-9) \rceil, \lceil (32-0)/(18-2) \rceil, \lceil (32-8)/(18-7) \rceil, \lceil (32-24)/(18-12) \rceil) = \max(3, 2, 2, 3, 2) = 3$
$IUSI_3 = \max(\lceil (26-2)/(16-4) \rceil, \lceil (26-14)/(16-9) \rceil, \lceil (26-0)/(16-2) \rceil, \lceil (26-8)/(16-7) \rceil, \lceil (26-22)/(16-12) \rceil) = \max(2, 2, 2, 2, 1) = 2$
$IUSI_4 = \max(\lceil (34-2)/(19-4) \rceil, \lceil (34-14)/(19-9) \rceil, \lceil (34-0)/(19-2) \rceil, \lceil (34-8)/(19-7) \rceil, \lceil (34-24)/(19-12) \rceil) = \max(3, 2, 2, 3, 2) = 3$
$IUSI_5 = \max(\lceil (36-2)/(21-4) \rceil, \lceil (36-14)/(21-9) \rceil, \lceil (36-0)/(21-2) \rceil, \lceil (36-8)/(21-7) \rceil, \lceil (36-24)/(21-12) \rceil) = \max(2, 2, 2, 2, 2) = 2$
$IUSI = \max(IUSI_1, IUSI_2, IUSI_3, IUSI_4, IUSI_5) = 3$

In conclusion, a minimum of 3 processing units is needed to find an optimal schedule. As will be seen in Section 5, the upper bound is a higher value. ∎

4 PERIODIC SCHEDULING AND THE LOWER BOUND

Periodic scheduling comes with a new problem: as each task has multiple instances, we need to deal with an unlimited number of task instances. The solution is to start from the periodic behavior of each task and to derive a periodic behavior of the overall system. Further adding to the already high complexity of the problem, each task has its own period, which is independent from the periods of the other tasks. The issue is then to find a general period of the system, which accounts for the periods of all tasks.

In [28], a general period of a set of periodic tasks has been proposed. As the task periods are independent and thus may have any values, the solution is to consider the least common multiple of these values. To do that, we need to be able to treat the task periods as integer numbers. Fortunately, this is always the case; if necessary, one can go down until the hardware level, where each program execution is carried out during an integer number of machine cycles. However, usually the least common multiple can be computed at higher levels.

Definition 4.1. *For a task set $\mathcal{T}$, we define the general period P as the lowest time interval with the property that $\forall i \in \{1, \ldots, n\}$, $\exists r_i \in \mathbb{N}^*$ such that $P = r_i \cdot p_i$.* ∎

The meaning of Definition 4.1 can be described this way: at time t, some of the tasks in the set could be executed (i.e., for such a task there is an instance whose start time comes before t and whose deadline falls after t), while other tasks cannot be executed. Then, at time $t + P$, all tasks will be in exactly the same situation as they were at time t. This does not necessarily mean, however, that precisely the same tasks will be running at time $t + P$ as they were at time t.

With the general period defined above, we start by extending the definitions used in the single-instance case to the periodic case. In doing this, we follow the method used in [28]. Also, in order to avoid further complication of the formulas, in the sequel we will use the following notations:

- The term $c_{i,k} = c_i$ will denote not just the computational cost, but the sum between the computational cost and the time required for the processing unit to move to the servicing location and them back to the base location (that is, $m_i \cdot 2$); this is correct, as this enhanced value shows the total amount of time required for a processing unit to service one node.
- Accordingly, the term $s_{i,k}$ will be used to denote the moment that the processing unit may start moving towards the servicing location, which is m_i before the start time of instance k of task T_i.
- In addition, the term $d_{i,k}$ will denote the moment when the processing unit must have arrived from the servicing location, which is m_i after the deadline of instance k of task T_i.

Definition 4.2. *For each deadline $d_{i,k}$, we define the requested execution time, denoted by $R_{i,k}$, as the minimum amount of processor time that is necessary until time $d_{i,k}$, such that all task instances can meet their deadlines.* ∎

The analytical expression of $R_{i,k}$ is as follows:

$$R_{i,k} = \sum_{j=1}^{n} \sum_{\substack{h \\ d_{j,h} \leq d_{i,k}}} c_{j,h} + \\ + \sum_{j=1}^{n} \sum_{\substack{h \\ d_{i,k} < d_{j,h} < d_{i,k} + c_{j,h}}} (c_{j,h} - (d_{j,h} - d_{i,k}))$$

Definition 4.3. *For each deadline $d_{i,k}$ and for each start time $s_{b,l}$, we define the available execution time, denoted by $A_{i,k}^{b,l}$, as the maximum amount of execution time that may occur before time $s_{b,l}$ for the processes that need to be executed, completely or partly, until time $d_{i,k}$.* ∎

As for the single-instance case, the set of tasks that contribute to the expression of $A_{i,k}^{b,l}$ is a subset of those who contribute to the expression of $R_{i,k}$. The analytical expression of $A_{i,k}^{b,l}$ is:

$$A_{i,k}^{b,l} = \sum_{j=1}^{n} \sum_{\substack{h \\ s_{j,h} + c_{j,h} \leq s_{b,l} \\ d_{j,h} \leq d_{i,k}}} c_{j,h} + \\ + \sum_{j=1}^{n} \sum_{\substack{h \\ s_{j,h} < s_{b,l} < s_{j,h} + c_{j,h} \\ d_{j,h} \leq d_{i,k}}} (s_{b,l} - s_{j,h}) + \\ + \sum_{j=1}^{n} \sum_{\substack{h \\ s_{j,h} + c_{j,h} \leq s_{b,l} \\ d_{i,k} < d_{j,h} < d_{i,k} + c_{j,h}}} (c_{j,h} - (d_{j,h} - d_{i,k})) + \\ + \sum_{j=1}^{n} \sum_{\substack{h \\ s_{j,h} < s_{b,l} < s_{j,h} + c_{j,h} \\ d_{i,k} < d_{j,h} < d_{i,k} + c_{j,h}}} \max(s_{b,l} - s_{j,h}, c_{j,h} - \\ - (d_{j,h} - d_{i,k}))$$

Definition 4.4. *Let IUP be the Improved Utilization of Periodic task set $\mathcal{T}$, computed as follows:*

$$IUP = \max_{d_{i,k} > s_{b,l}} \lceil (R_{i,k} - A_{i,k}^{b,l}) / (d_{i,k} - s_{b,l}) \rceil, \\ \forall i, b \in \{1, \ldots, n\}, k, l \in \mathbb{N}$$

∎

We also denote by $IUP(i, k)$ the partial computation of the IUP, restricted to the instances of all tasks with deadlines until $d_{i,k}$: $IUP(i,k) = \max_{d_{m,o} > s_{b,l}} \lceil (R_{m,o} - A^{b,l}_{i,k})/(d_{m,o} - s_{b,l}) \rceil$, $\forall m, o, d_{m,o} \leq d_{i,k}$

Theorem 4.1. *The lower bound on the number of processors, determined by the computation of the partial Improved Utilization of Periodic task set, has a periodic behavior, given by the general period P of the task set:* $\max_i IUP(i,k) = \max_i IUP(i, k+r_i)$.

Proof The proof is very much similar to the one given in [28].

We start by separating $R_{i,k}$ into the two terms that make its expression. When computing $R_{i,k+r_i}$, the first term increases by $\sum_{j=1}^{n} r_j \cdot c_j$; this is because at time $d_{i,k} + P$, for each task T_j, there will be exactly r_j additional instances that reached their deadlines, when compared to time $d_{i,k}$.

As for the second term, we look for the task instances $T_{j,h}$ for which $d_{i,k} < d_{j,h} \leq d_{i,k} + c_{j,h}$. There are two cases:

a) At time $d_{i,k}$ there is an instance h of task T_j with $d_{i,k} < d_{j,h} \leq d_{i,k} + c_{j,h}$, that is, a task satisfying the condition. Then, after one general period P, we have $d_{i,k+r_i} < d_{j,h+r_j} \leq d_{i,k+r_i} + c_{j,h+r_j}$, so $T_{j,h+r_j}$ also satisfies the condition.

b) At time $d_{i,k}$ there is an instance h of task T_j with $d_{j,h} \leq d_{i,k}$ and $d_{i,k} + c_{j,h} < d_{j,h+1}$, so no instance of task T_j to satisfy the condition. Then, after one general period P, we have $d_{j,h+r_j} \leq d_{i,k+r_i}$ and $d_{i,k+r_i} + c_{j,h+r_j} < d_{j,h+r_j+1}$, so there is no instance of task T_j to satisfy the condition.

By summing the two terms, we get $R_{i,k+r_i} = R_{i,k} + \sum_{j=1}^{n} r_j \cdot c_j$.

A similar reasoning applies to the four terms of $A^{b,l}_{i,k}$. The first term increases by $\sum_{j=1}^{n} r_j \cdot c_j$, because after one general period, for each task T_j, there will be exactly r_j additional instances that reached their start times and deadlines. As for the other three terms, as before, at each time t there may be at most one instance of a task T_j which satisfies the conditions $s_{j,h} < s_{b,l} < s_{j,h} + c_{j,h}$ or $d_{i,k} < d_{j,h} < d_{i,k} + c_{j,h}$ or both; then, at time $t + P$, the situation will be precisely the same. This leads to $A^{b,l+r_b}_{i,k+r_i} = A^{b,l}_{i,k} + \sum_{j=1}^{n} r_j \cdot c_j$, so $R_{i,k+r_i} - A^{b,l+r_b}_{i,k+r_i} = R_{i,k} - A^{b,l}_{i,k}$.

Now we consider a deadline and a start time from two different general periods, that is, $d_{i,k+r_i}$ and $s_{b,l}$. We then get: $max\lceil (R_{i,k+r_i} - A^{b,l}_{i,k+r_i})/(d_{i,k+r_i} - s_{b,l}) \rceil = max\lceil ((R_{i,k} - A^{b,l}_{i,k}) + \sum_{q=1}^{n} r_q \cdot c_q)/((d_{i,k} - s_{b,l}) + P) \rceil$

As shown in [28], this would increase the maximum value (the lower bound) throughout the general periods only if $max\lceil (R_{i,k} - A^{b,l}_{i,k})/(d_{i,k} - s_{b,l}) \rceil < (\sum_{q=1}^{n} r_q \cdot c_q)/P = \sum_{q=1}^{n} c_q/p_q$ for $k = \overline{0, r_i - 1}$, $i = \overline{1, n}$ (i.e., the first general period). This is not true, however, because *IUSI* has been particularly designed to provide enough processor time such that all tasks can meet their deadlines. Thus, when task instances from different general periods are considered, the lower bound does not increase.

In conclusion, $\max_i IUP(i,k) = \max_i IUP(i, k+r_i)$, so the lower bound can be determined by making the computation only for the first general period. ∎

5 THE UPPER BOUND

The upper bound is intended to provide a limit in the number of processors beyond which any further addition of new processors is useless. Any scheduling algorithm should be able to find a feasible schedule with this number of processors. While the aim is always to find a feasible schedule for the lowest number of processors possible, the upper bound is useful because it reveals the complexity of the problem at hand.

A measure of the upper bound has been proposed in [4] for non-preemptive, single-instance task sets. Just as for the lower bound, the result is also valid for preemptive task sets. It involves counting, for each moment, the tasks that may be in execution when that moment arrives. In fact, such counting remains constant during a certain period of time, until one or more tasks are beginning and/or terminating; in this context, beginning means reaching the start time, while terminating means reaching the deadline. Thus, these change moments are the only ones when counting needs to be performed. For cyber-physical systems, the difference is that the start time has to be replaced with the moment when the servicing of the task may begin, and the deadline is replaced with the moment when the servicing has to stop. This way, the delays introduced by the movement of the processing unit to the processing location and back to the base location are taken into account. As in the previous sections, one has to subtract the movement delay from the start time and to add it to the deadline.

The upper bound is then determined through the following steps:

- Build the set U of all moments when the servicing of at least one task may begin. As seen before, for each task T_i, the corresponding moment is time $s_i - m_i$.
- Add to the set U all moments when the servicing of at least one task has to end. For each task, this moment is the task deadline.
- U is a true set: if multiple change points fall in the same moment, that moment is considered only once.
- Sort U in ascending order.
- For each element $u_j \in U$, compute IUP_j, namely the number of tasks whose servicing may start no later than u_j and has to end no sooner than u_{j+1}.
- The upper bound is $IUP = \max_j (IUP_j)$.

Theorem 5.1. *In a system with IUP processors, it is always possible to find a feasible schedule for the given task set.*

Proof From the definition of IUP, at each moment t, the number of tasks that may be executed is less than or equal to IUP. Thus, at each moment t, there are enough processors available for all tasks that might need to be scheduled at that moment. Since this holds for all moments, the conclusion is that any scheduling algorithm can find a feasible schedule for IUP processors. ∎

Example 5.1. *Let us consider the same task set as in Example 3.1. We then get the following values for the moments recorded by* U:
$IUP_2 = 1$
$IUP_4 = 2$
$IUP_7 = 3$
$IUP_9 = 4$
$IUP_{12} = 4$
$IUP_{16} = 3$
$IUP_{18} = 2$
$IUP_{19} = 1$

Hence, the value of the upper bound is $IUP = \max_j(IUP_j) = 4$. *This value is less than the total number of tasks, but higher than the lower bound.* ∎

As shown in [28], the measure can be extended to periodic tasks and it exhibits periodic behavior with respect to the general period of the task set, as it was defined before. The same reasoning can be applied here, by simply replacing the start time of each task instance ($s_{i,k}$) with the moment when its servicing may begin ($s_{i,k} - m_i$). In conclusion, considering the first general period is enough to determine the upper bound.

6 CONCLUSION AND FUTURE WORK

This paper improves the previous work of the authors in determining the number of processors that allows finding a feasible schedule for a task set, if one exists. Both the single-instance case and the periodic case are discussed, and the relation between the two cases is determined. In addition, the technique is adapted for mobile cyber-physical systems, for which the delays introduced by the movement of the processing units to various locations must be considered.

The results obtained so far have been derived from the general characteristics of task scheduling. This is clearly an advantage, as it provides bounding estimates that are always valid, regardless of the scheduling algorithm, and a limitation, since no information about the scheduling algorithm is used. Further improvement could be achieved by following that direction, that is, taking into account the particular behavior of the various scheduling algorithms.

Another open problem is that, while the upper bound provides a clear-cut limit, the lower bound does not guarantee finding a feasible schedule. Thus, it would be useful to study how often the lower bound is enough in practice for achieving a feasible schedule and to determine whether there is a certain dependency between the lower bound and the number of processors that are actually needed for a given task set.

REFERENCES

[1] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, *Scheduling independent multiprocessor tasks*, Proceedings of the 5th Annual European Symposium of Algorithms, pp. 1–12, Berlin, 1997. Springer Verlag.

[2] Ş. Andrei, A.M.K. Cheng, G. Grigoraş, and V. Rădulescu. *An efficient scheduling algorithm for the non-preemptive independent multiprocessor platform*, International Journal of Grid and Utility Computing, 3(4):215–223, 2012.

[3] Ş. Andrei, A.M.K. Cheng, and V. Rădulescu, *Estimating the number of processors towards an efficient non-preemptive scheduling algorithm*, Proceedings of 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'11), pp. 93–100, IEEE Computer Society, Timisoara, 2011.

[4] Ş. Andrei, A.M.K. Cheng, and V. Rădulescu, *An Improved Upper-bound Algorithm for Non-preemptive Task Scheduling*, Proceedings of 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'15), pp. 153–159, IEEE Computer Society, Timisoara, 2015.

[5] S. K. Baruah, *The non-preemptive scheduling of periodic tasks upon multiprocessors*, Real-Time Syst., 32(1-2):9–20, 2006.

[6] J. Blazewicz, P. Dell'Olmo, M. Drozdowski, and M. G. Speranza, *Scheduling multiprocessor tasks on three dedicated processors*, Inf. Process. Lett., 41(5):275–280, 1992.

[7] G. C. Buttazzo, M. Bertogna, and G. Yao, *Limited Preemptive Scheduling for Real-Time Systems: a Survey*, IEEE Transactions on Industrial Informatics (Volume: 9, Issue: 1), pp. 3–15, ISSN: 1551–3203, 2013, IEEE Computer Society.

[8] Y. Cai and M. C. Kong, *Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems*, Algorithmica, 15(6):572–599, 1996.

[9] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, *A categorization of real-time multiprocessor scheduling problems and algorithms*, Handbook on Scheduling Algorithms, Methods, and Models. Chapman Hall/CRC, Boca, 2004.

[10] Y. Chao, S. Lin, K. Lin, *Schedulability issues for EDZL scheduling on real-time multiprocessor systems*, Information Processing Letters, 107(5):158–164, 2008.

[11] Cheng, Albert M. K., *Real-Time Systems: Scheduling, Analysis, and Verification*, John Wiley & Sons, Inc., 2002.

[12] R. I. Davis, A. Burns, *FPZL Schedulability Analysis*, 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 245–256, Chicago, 2011.

[13] R. I. Davis, S. Kato, *FPSL, FPCL and FPZL Schedulability Analysis*, Real-Time Systems, 48(6):750–788, 2012.

[14] M. L. Dertouzos, *Control robotics: The procedural control of physical processes*, Information Processing, 74:807–813, 1974.

[15] M. L. Dertouzos, A. K. Mok, *Multiprocessor online scheduling of hard-real-time tasks*, IEEE Transactions on Software Engineering, 15(12):1497–1506, 1989.

[16] S. Dolev and A. Keizelman, *Non-preemptive real-time scheduling of multimedia tasks*, Real-Time Syst., 17(1):23–39, 1999.

[17] M. Grenier and N. Navet, *Fine-tuning MAC-level protocols for optimized real-time qos*, IEEE Transactions on Industrial Informatics, vol. 4, no. 1, pp. 6Ű-15, 2008.

[18] N. Guan, W. Yi, Z. Gu, Q. Deng, and G. Yu, *New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms*, RTSS '08: Proceedings of the 2008 Real-Time Systems Symposium, pp. 137–146, Washington, DC, USA, 2008. IEEE Computer Society.

[19] J. A. Hoogeveen, S. L. van de Velde, and B. Veltman, *Complexity of scheduling multiprocessor tasks with prespecified processor allocations*, Discrete Appl. Math., 55(3):259–272, 1994.

[20] M. Kubale, *Preemptive scheduling of two-processor tasks on dedicated processors*, Automatyka, 1082:145–153, 1990.

[21] E. A. Lee, *Cyber Physical Systems: Design Challenges*, ISORC '08: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing pp. 363–369, 2008.

[22] S. K. Lee, *On-line multiprocessor scheduling algorithms for real-time tasks*, TENCON '94. IEEE Regions 10's Ninth Annual International Conference,

pp. 607–611, 1994.

[23] C. Liu, L. Zhang, D. Zhang, *Task Scheduling in Cyber-Physical Systems*, Ubiquitous Intelligence and Computing, 2014 IEEE 11th Intl. Conf. on and IEEE 11th Intl. Conf. on Autonomic and Trusted Computing, and IEEE 14th Intl. Conf. on Scalable Computing and Communications and Its Associated Workshops, pp. 319–324, 2014.

[24] R. Marau, P. Leite, M. Velasco, P. Marti, L. Almeida, P. Pedreiras, and J. Fuertes, *Performing flexible control on low-cost microcontrollers using a minimal real-time kernel*, IEEE Transactions on Industrial Informatics, vol. 4, no. 2, pp. 125Ű-133, 2008.

[25] A. K. Mok, *Fundamental design problems of distributed systems for the hard-real-time environment*, Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[26] S. Park, J. Kim, and G. Fox, *Effective real-time scheduling algorithm for cyber physical systems society*, Future Generation Computer Systems, vol. 32, pp. 253–259, 2014.

[27] H. Ramaprasad and F. Mueller, *Tightening the bounds on feasible preemptions*, ACM Transactions on Embedded Computing Systems, vol. 10, no. 2, pp. 1-Ű34, 2010.

[28] V. Rădulescu, Ş. Andrei, and A.M.K. Cheng, *Resource Bounding for Non-preemptive Task Scheduling on a Multiprocessor Platform*, Proceedings of 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'16), pp. 157–164, IEEE Computer Society, Timisoara, 2016.

[29] L. Sha, R. Rajkumar, and J. Lehoczky, *Priority inheritance protocols: An approach to real-time synchronization*, IEEE Transactions on Computers, vol. 39, no. 9, pp. 1175-Ű1185, 1990.

[30] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo *Implications of classical scheduling results for real-time systems*, Computer, 28(6):16–25, 1995.