On the complexity of optimal priority assignment for periodic tasks upon identical processors

Liliana Cucu

On the complexity of optimal priority assignment for periodic tasks upon identical processors

Liliana Cucu, LORIA-INPL
615 rue du Jardin Botanique
Villers-les-Nancy, France
liliana.cucu@loria.fr

Abstract

In this paper we study global fixed-priority scheduling of periodic task systems upon identical multiprocessor platforms. Based on existing feasibility tests for periodic task systems upon identical multiprocessor platforms, we show (using a dummy priority assignment algorithm) that optimal priority assignment for these systems exists. Then we provide an algorithm based on RM-US$[m/(3m-2)]$ that has lower complexity. Finally, we conjuncture that, contrary to the general opinion, (pseudo-) polynomial optimal priority assignment algorithms for periodic task systems upon identical processors might exist.

1 Introduction

Real-time systems are generally embedded and are interacting with the environment. Requests in real-time environment are often of a recurring nature. Such systems are typically modeled as finite collections of simple, highly repetitive tasks. When the different instances of those tasks are generated in a very predictable manner, we deal with periodic tasks. A periodic task τ_i generates jobs at each integer multiple of its period T_i with the restriction that the first job is released at time O_i (the task offset).

The real-time performances of periodic tasks on uniprocessor have been extensively studied since the seminal paper of Liu and Layland [7] which introduces a model of periodic systems. The literature considering scheduling algorithms and feasibility tests for uniprocessor scheduling is tremendous. In contrast for *multiprocessor* parallel machines the problem of meeting timing constraints is a relatively new research area.

In this work we deal with **global scheduling**. By global scheduling, we understand that *task migration* is allowed (i.e., different jobs of an individual task may execute upon different processors) as well as *job migration* (an individual job that is preempted may resume execution upon a processor different from the one upon which it had been executing prior to preemption).

We deal also with **identical processors**. By identical processors, we understand that all processors have the same computing power for all tasks.

The *scheduling algorithm* determines which job[s] should be executed at each time instant. When priorities are assigned to the tasks during the entire life of tasks, we have a *fixed-priority* scheduling algorithm. If there is at least one fixed-priority schedule satisfying all constraints of the system, then we say that there is at least a *feasible priority assignment*. A fixed-priority scheduling algorithm is *optimal* if the algorithm provides a feasible priority assignment, if any.

Related research. The problem of scheduling periodic task systems on several processors was originally studied in [6]. Recent studies provide a better understanding of that scheduling problem and provide first solutions. E.g., [3] presents a categorization of real-time multiprocessor scheduling problems.

Initial results indicate that real-time multiprocessor scheduling problems are typically not solved by applying straightforward extensions of techniques used for solving similar uniprocessor problems because of *scheduling anomalies* [5].

The main fixed-priority algorithm (in the uniprocessor case) Rate Monotonic (RM) is no longer optimal in the multiprocessor case and different versions of RM were proposed for the multiprocessor case [1]. Particular anomalies for fixed-priority algorithms were also underlined in [2], e.g., the priority assignment given by Audsley in the uniprocessor case is no longer optimal in the multiprocessor case. Moreover, to the best of our knowledge, the literature does not provide any optimal priority assignment algorithm for periodic task systems scheduled using preemption upon identical processors. This paper is a first step to fill this gap by using existing feasibility tests for periodic task systems upon identical processors [4].

Contribution of this paper In this paper we study global fixed-priority scheduling of periodic task systems upon identical multiprocessor platforms. First we propose a dummy algorithm belonging to $O(n!)$ that is based on existing feasibility tests for periodic task systems upon identical multiprocessor platforms. Thus, we show that optimal priority assignment for these systems exists. Then we provide an algorithm based on $\mathrm{RM\text{-}US}[m/(3m-2)]$ with lower complexity. Finally, we conjuncture that, contrary to the general opinion, (pseudo-)polynomial optimal priority assignment algorithms might exist.

Organization of the paper The paper is organized as follows. Section 2 introduces the model and the notations necessary to the understanding of the paper. Section 3 provides the main contribution of this paper and we conclude in Section 4.

2 Model and notations [4]

We consider the scheduling of periodic task systems. A system τ is composed by n periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$, each task is characterized by a period T_i, a relative deadline D_i, an execution requirement C_i and an offset O_i. Such a periodic task generates an infinite sequence of jobs, with the k^{th} job arriving at time-instant $O_i + (k-1)T_i$ $(k = 1, 2, \ldots)$, having an execution requirement of C_i units, and a deadline at time-instant $O_i + (k-1)T_i + D_i$.

We will distinguish between *implicit deadline* systems where $D_i = T_i, \forall i$; *constrained deadline* systems where $D_i \leq T_i, \forall i$ and *arbitrary deadline* systems where there is no relation between the deadlines and the periods.

We consider in this paper a discrete model i.e., the characteristics of the tasks and the time are integers. Moreover, we consider that task parallelism is forbidden: a task cannot be scheduled at the same instant on different processors.

All scheduling algorithms considered in this paper are *deterministic* and *work-conserving* with the following definitions given below.

Definition 1 (Deterministic algorithm)**.** *A scheduling algorithm is said to be* deterministic *if it generates a unique schedule for any given sets of jobs .*

Definition 2 (Work-conserving algorithm)**.** *A* work-conserving *algorithm is defined to be the one that never idles a processor while there is at least one active task.*

By default, we consider that all the fixed-priority schedulers for whom we provide the results in Section 3 are always deterministic and work-conserving.

3 Priority assignment

In this section we prove that optimal priority assignment algorithm for periodic systems (be they constrained, implicit or arbitrary deadline task systems) does exist in the sense that if there is at least one feasible priority assignment, then the algorithm will find it. We prove this property by proposing in Section 3.1 a dummy algorithm (of $n!$ complexity) which consider all possible sequences of priority assignment and test the feasibility of the task system. The feasibility issue is solved using existing feasibility tests given in [4]. These latter multiprocessor tests have a pseudo-polynomial complexity and they do not do worse than uniprocessor tests.

Finally in Section 3.2, we improve the complexity of the dummy algorithm by using a branch & bound algorithm. This algorithm is based on algorithm $\mathrm{RM\text{-}US}[m/(3m-2)]$ given in [1]. Moreover, we discuss the fact that worst-case behaviour of this algorithm is probably a rare event and one can use large deviations approaches to prove its complexity.

3.1 Optimal priority assignment

We consider a task system $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ of n periodic tasks with $\tau_i = (O_i, C_i, T_i, D_i)$. Task system τ can be an implicit, constrained or arbitrary deadline task system.

We define a working variable $\mathcal{W} \in \{1, 2, \cdots, n\}^n$ such that the i'th element of $\mathcal{W}$ is equal to $j \in \{1, 2, \cdots, n\}$ if and only if task τ_i has priority j. We consider that all tasks have different priorities, thus the i_1'th and the i_2'th elements of $\mathcal{W}$ are different if $i_1 \neq i_2$. For instance for a task system $\tau = \{\tau_1, \tau_3, \tau_2\}$ ordered from the highest priority task to the lowest priority task, we have $\mathcal{W} = (1, 3, 2)$.

Algorithm 1 Optimal priority assignment algorithm for periodic task upon identical parallel machines

```
Require: Task system $\tau$ and $m$ identical processors
Ensure: Priority assignment if it exists
 1: $\mathcal{W} := (1, 2, \cdots, n)$;
 2: $n_{tested_{config}} := 1$;
 3: $var_{Boolean} := false$;
 4: while $n \neq n!$ or $var_{Boolean} \neq true$ do
 5:     if Feasibility Test returns true then
 6:         $var_{Boolean} := true$;
 7:     else
 8:         $n_{tested_{config}} := n_{tested_{config}} + 1$;
 9:         increase $\mathcal{W}$;
10:     end if
11: end while
12: if $var_{Boolean} \neq false$ then
13:     There is no feasible priority assignment;
14: else
15:     $\mathcal{W}$ is a feasible priority assignment;
16: end if
```

In Algorithm 1, line 5, we use the feasibility test given in [4].

Theorem 1. *If a periodic task system τ is feasible under fixed-priority scheduling on m identical processors, then Algorithm 1 will find a feasible priority assignment.*

Proof. For any periodic task system there are $n!$ possible sequences of priority assignment. Given the condition $n \neq n!$ imposed in Algorithm 1, line 4 the algorithm tests all possible sequences unless it finds a feasible priority assignment. Thus, Algorithm 1 stops either if it finds a feasible priority assignment, or if it has visited all possible priority assignments and none of them is feasible. Given these two cases, we can conclude that Algorithm 1 will always find a feasible priority assignment, if it exists. □

Corollary 2. *For any periodic system τ that is feasible under fixed-priority scheduling on m identical processors, a feasible priority assignment can be found in $O(n!S)$, where S is the complexity of the feasibility test of a periodic task system under fixed-priority scheduling.*

Proof. The proof is obtained from the fact that Theorem 1 proves that Algorithm 1 is an optimal priority assignment algorithm that needs at most $n!$ steps to decide. □

3.2 Another priority assignment algorithm for implicit deadline tasks

In this section we improve the optimal priority assignment algorithm by proposing an algorithm of lower complexity, that stands only in the case of implicit deadline tasks.

The main idea of this algorithm comes from the observation given in [2] that "even if we could use schedulability tests that are necessary and sufficient, it is no longer possible to find an optimal priority assignment by using the *test for lowest priority viability approach*". This observation is based on the fact that exchanging the priorities between higher priority tasks can turn a schedulable system into an unschedulable one. Thus in the multiprocessor case, we cannot test a feasibility assignment starting from the lowest priority tasks to the highest ones. Therefore one maybe should do it from the highest priority tasks to the lowest ones. Algorithm 2 exploits this idea by starting to assign first higher priorities. Since the schedulability of higher priority tasks is not affected by lower priority tasks, we can test at each new step i the feasibility of the i tasks to whom priorities have been already assigned. Moreover, we exploit the feasibility result obtained for RM-US$[m/(3m-2)]$ for giving the highest priorities.

Require: Task system τ and m identical processors

Algorithm 2 Another priority assignment

```
Ensure: Priority assignment for task if it exists
 1: Choose a subset $\tau_0$ such that $m_0 = \min_{m_0=1,\cdots,m}\{U(\tau_0) \leq \frac{m^2}{3m-2}\}$ ;
 2: assign priorities for tasks belonging to $\tau_0$ according to RM-US[$m/(3m-2)$];
 3: $n_0 := n - card(\tau_0)$;
 4: $i_0 := n_0$;
 5: while $n_0 \neq 0$ do
 6:     assign priority $n_0$ to task ;
 7:     if FeasibilityTest returns true then
 8:         $n_0 := n_0 - 1$;
 9:         $i_0 := i_0 + 1$;
10:     else
11:         $i_0 := i_0 + 1$;
12:     end if
13: end while
```

Theorem 3. *Algorithm 2 is an optimal priority assignment for periodic tasks on m identical processors.*

Proof. Algorithm 2 is obviously optimal. □

Discussion on worst-case behaviour : We conjecture that Algorithm 2 behaves well in average situations and that the worst-case situations are rare events. To conclude on the average complexity of Algorithm 2 we need to use rare events theory since it is difficult to find representative task systems (large enough and random enough). If we can say how much worst-case complexity deviates from average complexity, this indicates that the following conjecture is true:

Conjecture 4. *There is an optimal priority assignment algorithm that has pseudo-polynomial complexity.*

4 Conclusions and future works

In this paper we prove that optimal priority assignment algorithm for periodic tasks upon identical processor does exist. The proposed dummy algorithm has $O(n!)$ complexity. We improve this complexity by giving a second algorithm. This second algorithm is based on RM-US[$m/(3m-2)$] algorithm. A first possible extension concerns the harmonic task systems and it can be obtained by replacing the latter algorithm with the RM-US$m/(2m-1)$ algorithm.

In order to conclude on the complexity of Algorithm 2, we are currently working on the evaluation of its performances on large set of tasks. We intend then to apply rare event theory to state on its average complexity. If we can apply this theory, we will obtain the proof that (pseudo-) polynomial optimal priority assignment algorithm for periodic tasks upon identical processors exists.

References

[1] A , B., B , S., J , J. Static-priority scheduling on multiprocessors. *Proceedings of the 22nd IEEE Real-Time Systems Symposium* (2001), 193–202.

[2] A , B., J , J. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. *Proceedings of the WIP session of IEEE Real-Time Systems Symposium (RTSS'00)* (2000), 53 – 56.

[3] C , J., F , S., H , P., S , A., A , J., B , S. A categorization of real-time multiprocessor scheduling problems and algorithms. *Handbook of Scheduling* (2005).

[4] C , L., G , J. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. *Proceedings of the 10th Design, Automation and Test in Europe (DATE'07)* (2007).

[5] D , S., L , C. On a real-time scheduling problem. *Operations Research(26)* (1978), 127–140.

[6] L , C. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary 37-60(II)* (1969), 28–31.

[7] L , C., L , J. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.