Scheduling Configuration on Posix 1003.1b Systems

Mathieu Grenier, Nicolas Navet

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Scheduling Configuration on Posix 1003.1b Systems

Mathieu Grenier — Nicolas Navet

N° 6209

June 2007

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6209--FR+ENG

INRIA LORRAINE

Scheduling Configuration on Posix 1003.1b Systems

Mathieu Grenier * , Nicolas Navet *

Thème COM — Systèmes communicants
Projet TRIO

Rapport de recherche n° 6209 — June 2007 — 26 pages

Abstract: Posix 1003.1b compliant systems provide two well-specified scheduling policies, namely *sched_rr* (Round-Robin like) and *sched_fifo* (FPP like). Up to now, little has been done to take advantage of the combination of both policies to maximize schedulability. In this study, we propose two priority, policy and quantum assignment algorithms for Posix 1003.1b systems that are optimal with regards to the power of the feasibility test (i.e. its ability to distinguish feasible and non feasible configurations). Though much less complex than an exhaustive exploration, the exponential complexity of the algorithms limits their applicability to small or medium-size problems. This is why a heuristic is proposed to handle larger task sets.

As shown in the experiments, our proposal allows achieving a significant gain in feasibility over FPP, and therefore using the computational resources at their fullest potential. In many cases where FPP does not lead to a feasible schedule, using both FPP and RR may provide an alternative to EDF, which, most often, is not implemented at the OS kernel level.

Key-words: Operating Systems, Multi-tasking, Scheduling, Round-Robin, Fixed-Priority Preemptive.

* LORIA - TRIO Group, Campus Scientifique - BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, France. Email: First-Name.LastName@loria.fr

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Configuration de l'ordonnancement sur les systèmes Posix 1003.1b

Résumé : Les systèmes conformes au standard Posix 1003.1b offrent deux politiques d'ordonnancement bien spécifiées, à savoir *sched_rr* et *sched_fifo*. Nous étudions dans ce rapport l'utilisation combinée de ces deux politiques dans l'optique de maximiser la faisabilité et proposons des algorithmes d'allocation de priorité, de politique d'ordonnancement et de quantum pour les systèmes Posix 1003.1b. Les algorithmes d'allocation sont montrés optimaux au regard du pouvoir du test de faisabilité, c'est-à-dire la capacité du test à distinguer les configurations faisables des configurations non-faisables. Bien que les algorithmes proposés réduisent la complexité du problème, celle-ci reste exponentielle en le nombre de tâches ce qui limite l'utilisation de ces algorithmes à des problèmes de petites et moyennes tailles. C'est pourquoi, nous proposons une heuristique peu complexe capable de traiter des problèmes de grandes tailles.

Par simulation, nous constatons que nos propositions permettent d'obtenir un gain significatif en faisabilité en comparaison à FPP. Dans de nombreuses configurations non-faisables avec FPP, l'utilisation combinée de FPP et RR permet de conduire à un ordonnancement faisable, et ainsi de fournir une alternative à EDF, politique qui, le plus souvent, n'est pas disponible dans les systèmes d'exploitation.

Mots-clés : Systèmes d'Exploitation, Multi-tâches, Ordonnancement, Round-Robin, Fixed-Priority Preemptive.

1 Introduction

Context of the study. This study deals with the scheduling of real-time systems implemented on Posix 1003.1b compliant Operating System (OS). Posix 1003.1b [7], previously known as Posix4, defines real-time extension to Posix mainly concerning signals, inter-process communications, memory mapped files, synchronous and asynchronous IO, timers and scheduling (a recap of Posix's features related to scheduling is given in §2.1). This standard has become very popular and most of today's OS conform, at least partially, to it.

Problem definition. Posix 1003.1b compliant OSs provide two scheduling policies, namely *sched_fifo* and *sched_rr*, which under some restrictions discussed in §2.1, are respectively equivalent to Fixed Preemptive Priority (FPP) and Round-Robin (RR for short). Thus, under Posix 1003.1b, each process is assigned both a priority, a scheduling policy and, in the case of Round-Robin, a quantum. At each point in time, one of the ready processes with the highest priority is executed, according to the rules of its scheduling policy (e.g., yielding the CPU after the end of a quantum under RR).

The problem addressed here is to assign priorities, policies and quanta to tasks in such a way as to respect deadline constraints. For FPP alone, the well-known Audsley algorithm [2] is optimal. Here we consider the cases (1) where the quantum value is a system-wide constant and (2) where quanta can be chosen on a task-per-task basis. At the time of writing, case (1) is under submission and case (2) has been published in [6]. As it will be seen in §3.2 and §4.2, the complexities of these two problems are such that an exhaustive search is not feasible even on small size problems. For instance, a task set of cardinality 10 with task-specific quanta chosen among 5 different values requires to analyze the feasibility of more than 10^{11} different configurations (see §4.2).

Contributions. Traditionally, the RR policy is only considered useful for low priority processes performing some background computation tasks "when nothing more important is running". In this study, as we did in [10, 6], we argue that the combined use of RR and FPP allows to successfully schedule a large number of systems that are unschedulable with FPP alone.

The first contribution consists of algorithms for assigning priorities, policies and quanta that are optimal in the sense that if there exists at least a feasible solution[1], then the algorithms will return one. The proposed algorithms are extensions of the classical Audsley algorithm [2]. Two algorithms are proposed. The first one, called *Audsley-RR-FPP*, handles the case where the quantum value is a system-wide constant. The next one [6], called *Audsley-RR-FPP**, extends *Audsley-RR-FPP* to take into account the case where the quanta are task-specific values. The worst-case complexity of the algorithms is assessed and a set of optimizations are proposed to reduce the search space. Despite the significant complexity reduction, the algorithms are limited to small or medium size problem. That is why we propose in addition an heuristic, called *Load-RR-FPP,* for large size problems in the case where the quantum value is a system-wide constant. The second contribution of the paper is that we give further evidences that the combined use of both FPP and RR is effective - especially when quanta can be chosen for each individual task - for finding feasible schedules even when the workload of the system is high.

Related work. We identify two closely related lines of research: schedulability analyses and priority assignment. Audsley in [2, 3] proposes an optimal priority assignment algorithm for FPP, that is now well-known in the literature as the Audsley algorithm. Later on in [5], this algorithm has been shown to be also optimal for the non-preemptive scheduling with fixed priorities. The problem of best assigning priorities and policies under Posix 1003.1b was first tackled in [9] but the solution relies on heuristics and is not optimal in the general case.

[1] We call here a *feasible* solution, a solution that successfully passes a schedulability test verifying property 2 (see §2.5). In the following, we make use of the response time bound analysis proposed in [9] to distinguish between feasible and non-feasible solutions.

The problem addressed here is different than in the plain FPP case because the use of RR leads to the occurrence of scheduling "anomalies", which are sometimes counter-intuitive. For instance, as it will be seen in §2.5, increasing the quantum value for a task can lead sometimes to a greater response time for this task. Similarly, decreasing the set of higher priority tasks, can increase the response time. This prevents us from using the proposed priority, policy and quantum assignment algorithm with the schedulability assessed by simulation, or with a feasibility test that would not possess some specific properties discussed in §2.5. Indeed there would be cases where the algorithm would discard schedulable assignments and thus not be optimal. In this study, feasibility is assessed by the analysis published in [9], which ensures that the computed response time bounds decrease when the set of higher priority tasks is reduced. This property enables us to use an Audsley-like algorithm for the assignment that will be shown to be optimal with regard to the power of the test, that is its ability to distinguish feasible or non feasible configurations.

Organisation. Section *2* summarizes the main features of the scheduling under Posix 1003.1b and introduces the model and notations. In sections 3 and 4, we present the two optimal priority, policy and quantum assignment algorithms: *Audsley-RR-FPP* (i.e., system-wide constant quantum) and *Audsley-RR-FPP** (i.e., quanta can be chosen on task-per-task basis). Then, an heuristic, called *Load-RR-FPP,* aimed at handling larger task sets is discussed in section 5. Efficiency of the proposals is then assessed in section 6.

2 Scheduling under Posix 1003.1b: model and basic properties

In this section we present the system model and summarize the main features related to scheduling of Posix 1003.1b. We then present the assumptions made in this study and derive some basic properties of the scheduling under Posix 1003.1b that will be used in the subsequent sections.

2.1 Overview of Posix 1003.1b scheduling

In the context of OS, we define a task as a recurrent activity which is either performed by repetitively launching a process or by a unique process that runs in cycle. Posix 1003.1b specifies 3 scheduling policies: *sched_rr*, *sched_fifo* and *sched_other*. These policies apply on a process-by-process basis: each process run with a particular scheduling policy and a given priority. Each process inherits its scheduling parameters from its father but may also change them at run-time.

- *sched_fifo* : fixed preemptive priority with First-In First-Out ordering among same-priority processes. In the rest of the paper, it will be assumed that all *sched_fifo* tasks of an application have different priorities. With this assumption and without change during run-time *sched_fifo* is equivalent to FPP.
- *sched_rr* : Round-Robin policy (RR) which allows processes of the same priority to share the CPU. Note that a process will not get the CPU until a higher priority ready-to-run processes are executed. The quantum value may be a system-wide constant, process specific or fixed for a given priority interval.
- *sched_other* is an implementation-defined scheduler. It could map onto *sched_fifo* or *sched_rr*, or also implement a classical Unix time-sharing policy. The standard merely mandates its presence and its documentation. Because we cannot rely on the same behaviour of *sched_other* under all Posix compliant OSs, it is strongly suggested not to use it if a portability is a matter of concern. We will not consider it in our analysis.

Associated with each policy is a priority range. Depending on the implementation, these priority ranges may or may not overlap but most implementations allow overlapping. Note that these previously explained scheduling mechanisms similarly apply to Posix threads with the system contention scope as standardised by Posix 1003.1c standard [7].

2.2 System model

The activities of the system are modeled by a set $\mathcal{T}$ of n periodic and independent tasks $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task τ_i is characterized by a tuple (C_i, T_i, D_i) where each request of τ_i, called an instance, has an execution time of C_i, a relative deadline D_i and a period equal to T_i time units. One denotes by $\tau_{i,j}$ the j^{th} release of τ_i. As usual, the response time of an instance is the time elapsed between its arrival and its end of execution.

Under Posix 1003.1b, see §2.1, each task τ_i possesses both a priority p_i and a scheduling policy $sched_i$. In this study, we choose the convention "the smaller the numerical value, the higher the priority". In addition to the priority, under RR, each task τ_i is assigned a quantum value ψ_i. The priority and scheduling policy assignment $\mathcal{P}$ is fully defined by a set of n tuples $(\tau_i, p_i, sched_i^{\mathcal{P}})$ (i.e. one for each task). A quantum assignment under $\mathcal{P}$, denoted by $\Psi_{\mathcal{P}}$, defines the set of quantum values $\psi_i^{\Psi_{\mathcal{P}}}$ where $\psi_i^{\Psi_{\mathcal{P}}}$ is the quantum of τ_i. The whole scheduling is fully defined by the tuple $(\mathcal{P}, \Psi_{\mathcal{P}})$ which is called a *configuration* of the system.

Under assignment $\mathcal{P}$, the set of tasks $\mathcal{T}$ is partitioned into separate layers, one layer for each priority level j where the layer $\mathcal{T}_j^{\mathcal{P}}$ is the subset of tasks assigned to priority level j. Under $\mathcal{P}$, $\mathcal{T}_{hp(j)}^{\mathcal{P}}$ (resp. $\mathcal{T}_{lp(j)}^{\mathcal{P}}$) denotes the set of all tasks possessing a higher (resp. lower) priority than j. A layer in which all tasks are scheduled with RR (resp. FPP) is called an RR layer (resp. FPP layer). In the following, $\mathcal{P}$ or $\Psi_{\mathcal{P}}$ will be omitted when no confusions are possible. A list of the notations is provided in appendix at the end of this report.

In the following, a task τ_i is said *schedulable* under assignment $(\mathcal{P}, \Psi_{\mathcal{P}})$ if its response time bound, as computed by the existing Posix 1003.1b schedulability analysis [9], is no greater than its relative deadline (i.e. maximum duration allowed between the arrival of an instance and its end of execution). The whole system is said schedulable if all tasks are schedulable. Note that the test presented in [9] is sufficient but not necessary, there are thus task sets which will not be classified as schedulable while there exist configurations under which no deadlines are missed.

2.3 Assumptions

In this study, as explained in *§2.1*, only *sched_fifo* and *sched_rr* are considered for portability concern. Due to the complexity of assigning priorities, quanta and scheduling policies, the following restrictions are made:

1. context switch latencies are neglected, but they could be included in the schedulability analysis of [9] as classically done (see, for instance, [11]),
2. since a priority level without any tasks has no effect on the scheduling, we impose the priority range to be contiguous,
3. two tasks having different scheduling policies have different priorities, i.e., $\forall i \neq j$, $sched_i \neq sched_j \implies p_i \neq p_j$,
4. all *sched_fifo* tasks must possess distinct priorities ($sched_i = sched_j = sched\_fifo \implies p_i \neq p_j$). With this assumption and without priority change at run-time, *sched_fifo* is equivalent to fixed-preemptive priority (FPP). Thus, several tasks having the same priority are necessarily scheduled under *sched_rr* policy,
5. the quantum value can be a system-wide constant or specific to each task.

2.4 Schedulability analysis under Posix: a recap [9]

In this paragraph, we summarize the schedulability analysis [9] of a configuration $(\mathcal{P}, \Psi_{\mathcal{P}})$ under Posix. Tasks scheduled under Posix can be described as a superposition of priority layers [9]. At each point in time, one of the ready instances with the highest priority (let's say p_i) is executed as soon as and as long as no instances in the higher priority layers (instances of tasks in $\mathcal{T}_{hp(p_i)}$) are pending. Inside each priority layer, instances are

scheduled either according to FPP or RR with the restrictions that all instances belonging to the same layer have the same policy.

FPP policy is achieved when a ready instance $\tau_{i,j}$ is executed when no higher priority instances is pending. Under RR, a task τ_i has repeatedly the opportunity to execute during a time slot of maximal length $\psi_i^{\Psi_\mathcal{P}}$. If the task has no pending instance or less pending work than the slot is long, then the rest of the slot is lost and the task has to wait for the next cycle to resume. The time between two consecutive opportunities to execute may vary, depending on the actual demand of the others tasks, but it is bounded by $\overline{\psi}_i^{\Psi_\mathcal{P}} = \sum_{\tau_k \in \mathcal{T}_{p_i}^\mathcal{P}} \psi_k^{\Psi_\mathcal{P}}$ in any interval where the considered task has pending instances at any moment. In [9], worst-case response time bounds for priority layers have been derived in a way that is independent from the scheduling policies used for each layer. This analysis is based on the concept of majorizing work arrival functions, which measure a bound on the processor demand, for each task, over an interval starting at a "generalized critical instant". The majorizing work arrival function on an interval of length t for a periodic task τ_i is:

$$s_i(t) = C_i \cdot \left\lceil \frac{t}{T_i} \right\rceil . \tag{1}$$

The worst-case response time bound can be expressed as

$$\max_{j<j^*}(e_{i,j} - a_{i,j}) \ , \tag{2}$$

where $j^* = \min\{j \mid e_{i,j} \leq a_{i,j+1}\}$, where $a_{i,j}$ is the release of the j^{th} instance of τ_i after the critical instant and $e_{i,j}$ is a bound on the execution end of this instance. Since τ_i is a periodic task, $a_{i,j} = (j-1) \cdot T_i$ $(j = 1, 2, ...)$. If τ_i is in an FPP layer, then

$$e_{i,j} = \min\{t > 0 \mid \widetilde{s}_i(t) + s_{i,j} = t\} \ , \tag{3}$$

where $\widetilde{s}_i(t) = \sum_{\tau_k \in \mathcal{T}_{hp(p_i)}^\mathcal{P}} s_k(t)$ is the demand from higher priority tasks (i.e. task in $\mathcal{T}_{hp(p_i)}^\mathcal{P}$) and $s_{i,j} = \sum_{i=1}^{j} C_i$ is the demand from previous instances and the current instance of τ_i. If τ_i is in an RR layer, then

$$e_{i,j} = \min\{t > 0 \mid \overline{\Psi}_i(t) + s_{i,j} = t\} \ , \tag{4}$$

where the demand from higher priority tasks and of all other tasks of the RR layer is:

$$\overline{\Psi}_i(t) = \min\left(\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_\mathcal{P}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_\mathcal{P}} - \psi_i^{\Psi_\mathcal{P}}) + \widetilde{s}_i(t), s_i^*(t) \right) \ , \tag{5}$$

where $\overline{\psi}_i^{\Psi_\mathcal{P}} - \psi_i^{\Psi_\mathcal{P}}$ is the sum of the quanta of all other tasks of the RR layer and

$$s_i^*(x) = \max_{u \geq 0} \left(s_i(u) + \widetilde{s}_i(u+x) + \overline{s}_i(u+x) - u \right), \tag{6}$$

where $\overline{s}_i(u+x) = \sum_{\tau_k \in \mathcal{T}_{p_i}^\mathcal{P} \setminus \{\tau_i\}} s_k(u+x)$ is the demand from other tasks than τ_i in $\mathcal{T}_{p_i}^\mathcal{P}$. The algorithm for computing the worst-case response time bounds can be found in [9]. It is to stress that this schedulability analysis is sufficient but not necessary; some task sets may fail the test while they are perfectly schedulable. This will certainly induce conservative results but the approach developed here remains valid with another - better - schedulability test as long as it is sufficient and possesses the properties described in §2.5.

2.5 Scheduling under Posix 1003.1b: basic properties

Under FPP, as well as under RR, any higher priority task will preempt a lower priority task thus the following properties hold for any task τ_i :

1. all ready instances, with higher priorities than p_i, will delay the end-of-execution of the instances of τ_i. It is worth noting that this delay is not dependent on the relative priority ordering among these higher priority instances and their quantum values,
2. lower priority instances, whatever their policy, will not interfer with the execution of instances of τ_i and thus won't delay their end-of-execution.

These two properties ensure that the following lemma, which is well-known in the FPP case, holds.

Lemma 1 *[3] The worst-case response time of an instance of τ_i only depends on the set of same priority tasks, the values of their quantum and the set of higher priority tasks. The relative priority order among higher priority tasks and the values of their quantum has no influence.*

However, despite lemma 1 holding, scheduling under RR leads to scheduling anomalies. Indeed, scheduling under Posix is often counter-intuitive. For instance, it has been shown in [4], that early end-of-executions can lead to missed deadlines in configurations that would be feasible with WCETs. Here, we highlight that increasing the quantum size of a task can increase its response time. Figures 1 and 2 present the scheduling of task set $\mathcal{T} = \{\tau_1, \tau_2\}$ where $\tau_1 = (C_1 = 2, T_1 = 5)$ and $\tau_2 = (4, 10)$. All the tasks belong to the same layer and the chosen quantum assignments are $\Psi = \{\psi_1 = 2, \psi_2 = 2\}$ (figure 1) and $\Psi' = \{\psi_1 = 2, \psi_2 = 3\}$ (figure 2) .

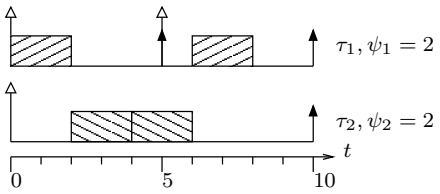


Figure 1: Scheduling of task set $\mathcal{T} = \{\tau_1, \tau_2\}$ with Round-Robin and quantum assignment $\Psi = \{\psi_1 = 2, \psi_2 = 2\}$.

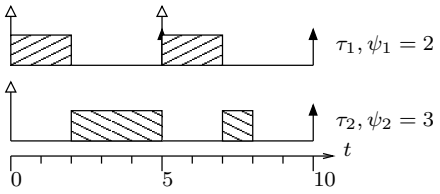


Figure 2: Scheduling of task set $\mathcal{T} = \{\tau_1, \tau_2\}$ with Round-Robin and quantum assignment $\Psi' = \{\psi_1 = 2, \psi_2 = 3\}$.

As it can be seen on figures 1 and 2, surprisingly the response time of τ_2 is 6 with a quantum of 2 and 8 with 3. However, with the schedulability analysis used in this study, property 1 holds and will be used to restrain the search space in section 4. A proof is given in appendix A.

Property 1 *Let τ_i be a task in a RR layer, increasing (resp. reducing) its quantum value, while reducing (resp. increasing) the quantum value of the other tasks of its RR layer, diminishes (resp. increases) the response time bound of τ_i computed with the chosen schedulability analysis.*

Furthermore, we would like to highlight that, quite surprisingly, reducing the set of higher or/and same priority tasks of a task does not necessarily diminish or leave unchanged its response time. The scheduling of task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ where $\tau_1 = (C_1 = 1, T_1 = 5, D_1 = 5)$, $\tau_2 = (1, 5, 5)$ and $\tau_3 = (6, 10, 10)$ with a system-wide quantum of 3 is presented under the assignments:

- $\mathcal{P} = \{(\tau_1, 1, sched\_fifo), (\tau_2, 2, sched\_rr), (\tau_3, 2, sched\_rr)\}$ on figure 3,

- $\mathcal{Q} = \{(\tau_1, 2, sched\_fifo), (\tau_2, 1, sched\_rr), (\tau_3, 1, sched\_rr)\}$ on figure 4.

As it can be seen on figures 3 and 4, the worst-case response time of τ_2 is 4 under $\mathcal{P}$ while it is 5 under $\mathcal{Q}$.

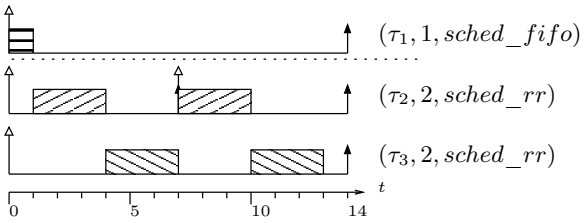


Figure 3: Scheduling of task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ where priorities $p_1 = 1$, $p_2 = p_3 = 2$ and policies $sched_1 = sched\_fifo$, $sched_2 = sched_3 = sched\_rr$ (assignment $\mathcal{P}$).

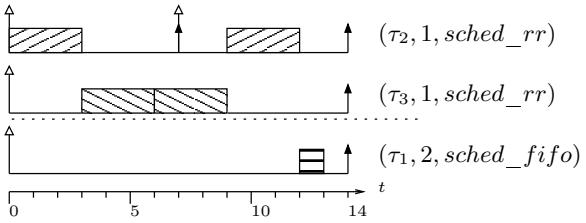


Figure 4: Scheduling of task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ where priorities $p_2 = p_3 = 1$, $p_1 = 2$ and where policies $sched_2 = sched_3 = sched\_rr$, $sched_1 = sched\_fifo$ (assignment $\mathcal{Q}$).

However, using response time bound analysis proposed in [9] ensures that property 2 holds.

Property 2 *Let τ_i be a task in RR or FPP layer, reducing its set of higher and same priority tasks, while keeping the quantum allocation unchanged within its Round-Robin layer (if τ_i is scheduled under RR), diminishes or leaves unchanged the response time bound of τ_i computed with the chosen schedulability analysis.*

Appendix B shows that the conservative response time bound computed with [9] ensures that property 2 holds. As it will be shown in appendix C, a schedulability test which ensures that property 2 is verified, allows to use an extension of the Audsley algorithm (as done in sections 3 and 4) and preserves its optimality with regards to the ability of the test to distinguish between feasible and non-feasible solutions (i.e., what is called the power of the test in the following).

3 Optimal assignment algorithm with system-wide quantum

We present an optimal priority and scheduling policy assignment algorithm when the quantum value is a system-wide constant[2]. This assignment is optimal when the schedulability analysis used verifies property 2 stated in §2.5. The optimality of this algorithm is proved in appendix C. It is worth pointing out that the policy is implied by the number of tasks having the same priority level: should only one task be assigned priority level i then its policy is FPP (i.e. a RR layer of cardinality 1 is strictly equivalent to an FPP layer, see §2.1), otherwise the policy is necessarily RR. The problem is thus reduced to the assignment of priorities since the quantum size is a system wide value. Our approach is an extension of the well-known Audsley algorithm [2], thus we call it the *Audsley-RR-FPP* algorithm.

[2]Since the quantum is here a system-wide constant, the notations for individualizing the quantum values are omitted in this section.

3.1 *Audsley-RR-FPP* algorithm

In the same way as the Audsley algorithm (abridged by AA in the following), the priorities are assigned from the lowest priority n to the highest priority 1 (line 3 in algorithm 1). The difference with AA lies in the fact that, at each priority level, the algorithm is not looking for a single task, but for a set of tasks (line 5 in algorithm 1).

Rationale of the algorithm. The main idea is to move, when needed, the maximum amount of workload to the lower priority levels and to schedule the corresponding tasks under RR. When an instance $\tau_{i,j}$ is assigned the same priority as $\tau_{k,h}$ and both are scheduled under RR, $\tau_{i,j}$ can delay $\tau_{k,h}$ less than if $\tau_{i,j}$ would be scheduled with a higher priority. The same argument holds for the delay induced by $\tau_{k,h}$ to $\tau_{i,j}$. Thus, as illustrated with an example in [10], task sets that are not feasible under FPP alone, become feasible with RR. Of course, in the general case, combining the use of both policies is the most efficient and, as it will be shown, leads to an optimal priority and policy assignment.

Input: task set $\mathcal{T} = \{\tau_1, \tau_2..., \tau_n\}$
Result: schedulable priority, scheduling policy assignment $\mathcal{P}$
Data: i: priority level to assign
$\mathcal{R}$: task-set with no assigned priority
$\mathcal{P}$: partial priority and policy assignment

```
1  R = T;
2  P = ∅;
3  for i = n to 1 do
4  |  try to assign priority i:
5  |  search a schedulable subset of tasks T_i in R
6  |  if no subset T_i is schedulable at priority i then
7  |  |  failure, return partial assignement:
8  |  |  return P;
9  |  else
10 |  |  let T_i a schedulable subset at priority i;
11 |  |  assign priority, policy and quantum:
12 |  |  if #T_i = 1 then
13 |  |  |  P = P ∪ {(τ_k, i, sched_fifo)}_{τ_k ∈ T_i};
14 |  |  else
15 |  |  |  P = P ∪ {(τ_k, i, sched_rr)}_{τ_k ∈ T_i};
16 |  |  end
17 |  |  remove T_i from R:
18 |  |  R = R\T_i;
19 |  end
20 |  if R = ∅ then return P;
21 end
```

Algorithm 1: *Audsley-RR-FPP* algorithm with a system-wide quantum value.

Step of the algorithm. For each priority level i, *Audsley-RR-FPP* algorithm attempts to find a schedulable subset $\mathcal{T}_i$ in the subset $\mathcal{R}$ (line 5 in algorithm 1) where $\mathcal{R}$ is made of all the tasks that have not been yet assigned a priority and a policy. The algorithm tries all possible subsets in $\mathcal{R}$, one by one, until a schedulable one is obtained or all possible assignments have been considered. In the latter case, the system is not schedulable (lines $7-8$ in algorithm 1). Otherwise, a schedulable subset $\mathcal{T}_i$ has been found. Precisely, $\mathcal{T}_i$ is schedulable when all tasks of $\mathcal{T}_i$ are feasible at priority i (scheduled under RR if $\#\mathcal{T}_i > 1$ or under FPP otherwise) while all tasks with no assignment (i.e., tasks in $\mathcal{R}\backslash\mathcal{T}_i$) have a priority greater than i. At each step, at least one task is assigned a priority and a policy (lines 11 to 16 in algorithm 1). Note that, when RR is used at least once, less than n priority levels are needed (early exit on line 20 in algorithm 1).

Looking for the set of schedulable tasks $\mathcal{T}_i$. There are $2^{\#\mathcal{R}}$ possible subsets $\mathcal{T}_i$ of $\mathcal{R}$ that can be assigned priority level i (line 5 in algorithm 1). A binary tree structure reflects the priority choices and the search for the schedulable subset is performed by exploring the tree. The basic exhaustive search is explained below. A method that speeds up the search by pruning away sub-trees that cannot contain a solution is provided in §3.2. As an illustration, figure 5 shows the search tree corresponding to set $\mathcal{R} = \{\tau_1, \tau_2, \tau_3\}$. Each edge is labeled either with "$= i$" (i.e., priority equal to i) or "$> i$" (i.e., priority greater than i). A label "$= i$" (resp. "$> i$") on the edge between vertices of depth k and $k+1$ means that the $(k+1)^{th}$ task of $\mathcal{R}$ belongs to the layer of priority i (resp. belongs to a layer of priority greater than i). Thus, a vertex of depth k models the choices performed for the k first tasks of $\mathcal{R}$. For instance, on figure 5, the vertex e implies that tasks τ_1 belongs to layer of priority i while task τ_2 does not. Each leaf is a complete assignment for priority level i, for instance leaf g corresponds to set $\mathcal{T}_i = \{\tau_1, \tau_2\}$.

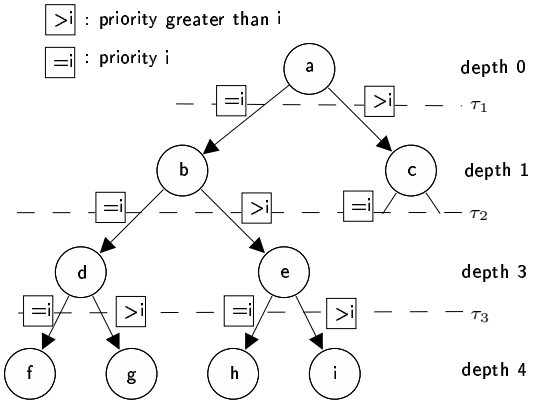


Figure 5: Search tree constructed in the search of a feasible subset of $\mathcal{R} = \{\tau_1, \tau_2, \tau_3\}$ at priority i. For instance, node b models the partial priority assignment where τ_1 is assigned priority i while node c means that τ_1 is assigned a greater priority.

The search is performed according to a depth-first strategy. The algorithm considers the first child of a vertex that appears and goes deeper and deeper until a leaf is reached, *i.e.*, until the set $\mathcal{T}_i$ is fully defined. When a leaf is reached, the schedulability of $\mathcal{T}_i$ is assessed. If $\mathcal{T}_i$ is feasible, the algorithm returns, otherwise, it backtracks till the first vertex such that not all its child vertices have been explored.

3.2 Complexity and improvements

Size of the search space. From [10], we know that in the worst-case an assignment with an exhaustive search implies testing $\sum_{k=1}^{n}\sum_{i=0}^{k}(-1)^{(k-i)}\binom{k}{i}i^n$ distinct scheduling configurations (e.g., $\approx 10^8$ configurations with 10 tasks). As it can be seen on figure 6, the space of search grows more than exponentially with the number of tasks, thus an exhaustive search is not feasible in practice in a wide range of real-time systems.

Audsley-RR-FPP. The algorithm looks at each priority level i for a subset of tasks $\mathcal{T}_i$ in the set $\mathcal{R}$ which are schedulable at priority i (line 5 in algorithm 1). At each priority level, there is $2^{\#\mathcal{R}}$ different possible subsets $\mathcal{T}_i$ in $\mathcal{R}$. Since at least one task is assigned to each priority level (otherwise the assignment would be non-schedulable, see line 8 in algorithm 1), the number of tasks belonging to $\mathcal{R}$ is lower than or equal to $\#\mathcal{T} - (n-i) = i$ when the assignment for priority level i is considered. Thus, the algorithm looks for $\sum_{i=1}^{n} 2^i = 2^{n+1} - 2$ assignments in the worst case. This complexity for a varying number of tasks is shown on figure 6, for instance, for a set of 10 tasks it is equal to 2046. Figure 6 shows also the size of the search space. Although a great complexity reduction with regards to a brute-force exhaustive search is achieved, the complexity remains exponential in the number of tasks. Thus, in practice, the proposal is not suited for large-size task sets that would, for instance, be

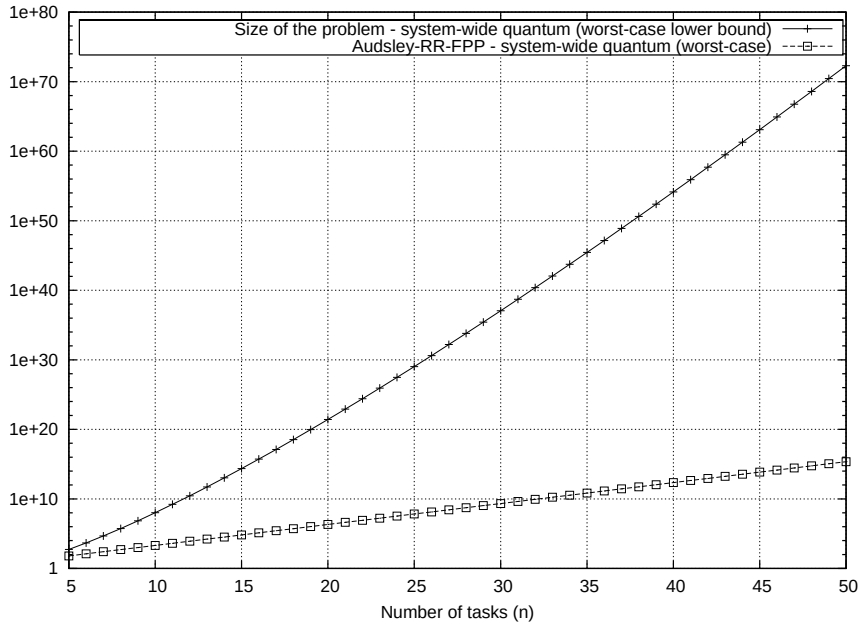


Figure 6: Complexity of the problem for a number of tasks varying from 5 to 50 (log. scale for y-axis) when the quantum is a system-wide value.

better handled by heuristics guiding the search towards promising parts of the search space. One such heuristic is proposed in section 5.

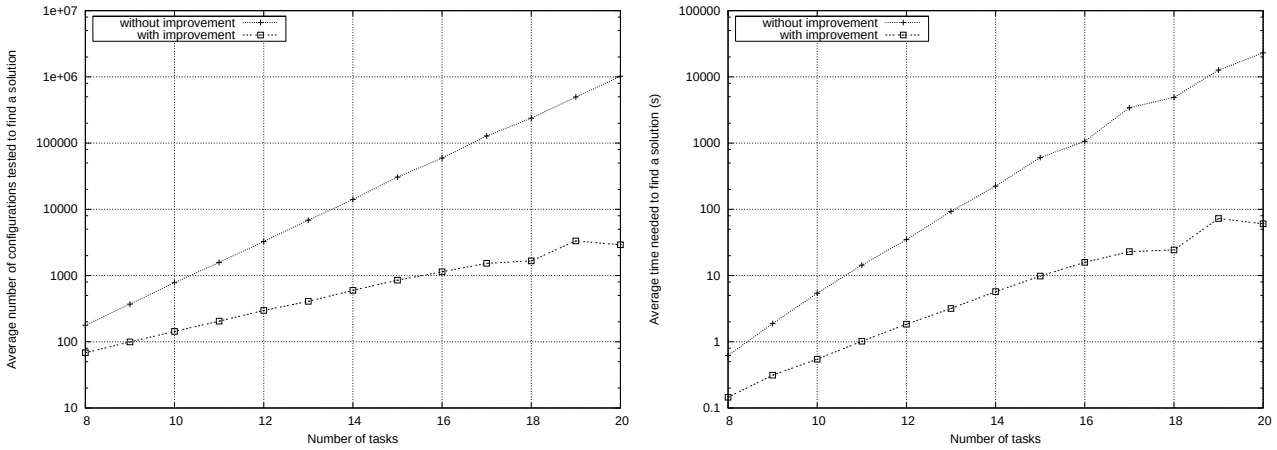
Complexity reduction. As seen before, the *Audsley-RR-FPP* performs an exhaustive search for each priority level i. To a certain extent, it is possible to reduce the number of sets that are to be considered. Indeed, thanks to property 2, one can identify and prune away branches of the search tree which necessarily lead to subsets $\mathcal{T}_i$ that are not schedulable.

With the basic algorithm explains in §3.1, feasibility is assessed at the leafs when all tasks have been given a priority. The idea is here to evaluate feasibility at intermediate vertices as well, by assigning a priority lower than i to the tasks for which no priority choice has been made yet. Under that configuration, if a task τ_i which is assigned the priority i is not schedulable, there is no need to consider the children of this vertex. Indeed, from property 2, since the priority assignment of the children of this vertex will increase the set of same or higher priority tasks, the response time of τ_i cannot decrease. Thus, all child vertices corresponds to priority assignments that are not schedulable.

The finding of this paragraph allows a very significant decrease in the average number of configurations tested by the *Audsley-RR-FPP* algorithm. For instance[3], as can be seen on figure 7(a), the average number of configurations that need to be tested with our method for 20 tasks is about 300 times smaller than the average number without this improvement. It is also noteworthy that the improvement ratio increases with the number of tasks (from 3 to 300 here). Logically, as can be seen on figure 7(b), the running time follows the same trend since it is directly proportional to the number of configuration tested: with the complexity pruning method, the running time ranges from 0.145s for 8 tasks to 60s for 20 tasks on an Intel Core Duo 1.8GHz.

The limit of the proposal is that, even with the complexity reduction procedure, the complexity increases exponentially with the number of tasks. Further experiments show that task sets of cardinality 30 to 35 can be handled in a few hours. Above this threshold, the running time will raise problem on certain task sets and the

[3]The experimental setup is given in section 6.



(a) (b)

Figure 7: Average number of configurations tested to find a solution and average running time on an Intel Core Duo 1.8GHz with and without the complexity reduction method. The CPU load U is equal to 0.84 with a number of tasks ranging from 8 to 20. Each point is the average value over 100 experiments and log. scale is used for y-axes.

algorithm is no more suited. Larger task sets can be treated with heuristics at the expense that results are no more optimal, see section 5.

4 Optimal assignment algorithm with task-specific quanta

We extend the algorithm proposed in section 3 to the case where quanta can be chosen on a task-per-task basis in the interval $[\psi_{\min}, \psi_{\max}]$, where $\psi_{\min}$ and $\psi_{\max}$ are natural numbers whose values are OS-specific constraints or chosen by the application designer. The corresponding algorithm is named *Audsley-RR-FPP**. This new algorithm remains optimal, see appendix C, when the feasibility is assessed with schedulability analysis verifying property 2 described in §2.5. With the assumptions made in section 2, the policy is implied by the number of tasks having the same priority level: should only one task be assigned priority level i then its policy is FPP (i.e. a RR layer of cardinality 1 is strictly equivalent to an FPP layer, see §2.1), otherwise the policy is necessarily RR. The problem is thus reduced to assigning priorities, plus quanta to tasks that belong to a RR layer.

4.1 *Audsley-RR-FPP** algorithm

In the same way as the *Audsley-RR-FPP* algorithm, the algorithm looks for a schedulable set of tasks at each priority level. However, since the quantum can be chosen, the algorithm examines, for each possible set of tasks, all possible quantum assignments until it finds a suitable one (line 5 in algorithm 2).

Step of the algorithm. For each priority level i (line 3 in algorithm 2), the *Audsley-RR-FPP** algorithm attempts to find a schedulable subset $\mathcal{T}_i$ in subset $\mathcal{R}$ (line 5 in algorithm 2) where $\mathcal{R}$ is made of all the tasks which have not been yet assigned a priority, a policy and a quantum. The algorithm tries all possible subsets of $\mathcal{R}$, one by one, and all possible quantum assignments for each subset until a schedulable configuration is obtained or all configurations have been considered. In the latter case, the system is not schedulable (lines 7 and 8 in algorithm 2). Otherwise, we have found a schedulable subset, denoted by $\mathcal{T}_i$, which, in the RR case, possesses quantum assignment $\{\psi_k\}_{\tau_k \in \mathcal{T}_i}$ (line 10 in algorithm 2). Precisely, $\mathcal{T}_i$ is schedulable when all tasks of $\mathcal{T}_i$ are feasible at priority i while all tasks without assignment (i.e., tasks in $\mathcal{R} \backslash \mathcal{T}_i$) have a priority greater than

Input: task set $\mathcal{T} = \{\tau_1, \tau_2..., \tau_n\}$
Result: schedulable priority, scheduling policy and quantum assignment $\mathcal{P}_k = (\mathcal{P}, \Psi_k)$
Data: i: priority level to assign
$\mathcal{R}$: task-set with no assigned priority
$\mathcal{P}$: partial priority and policy assignment
$\Psi_{\mathcal{P}}$: partial quantum allocation

```
1  R = T;
2  P = ∅;
3  for i = n to 1 do
4  |  try to assign priority i:
5  |  search a schedulable subset of tasks T_i under quantum allocation {ψ_k}_{τ_k∈T_i} in R
6  |  if no subset T_i is schedulable at priority i then
7  |  |  failure, return partial assignement:
8  |  |  return (P, Ψ_P);
9  |  else
10 |  |  let T_i a schedulable subset at priority i with quantum allocation {ψ_k}_{τ_k∈T_i};
11 |  |  assign priority, policy and quantum:
12 |  |  if #T_i = 1 then
13 |  |  |  P = P ∪ {(τ_k, i, sched_fifo)}_{τ_k∈T_i};
14 |  |  else
15 |  |  |  P = P ∪ {(τ_k, i, sched_rr)}_{τ_k∈T_i};
16 |  |  |  Ψ_P = Ψ_P ∪ {ψ_k}_{τ_k∈T_i};
17 |  |  end
18 |  |  remove T_i from R:
19 |  |  R = R\T_i;
20 |  end
21 |  if R = ∅ then return (P, Ψ_P);
22 end
```

Algorithm 2: *Audsley-RR-FPP** algorithm with task-specific quantum.

i. At each step, at least one task is assigned a priority and a policy (lines 11 to 17 in algorithm 2). Note that, when RR is used at least once, less than n priority levels are needed (early exit on line 21 in algorithm 2).

Looking for the set of schedulable tasks $\mathcal{T}_i$. There are $2^{\#\mathcal{R}}$ possible subsets $\mathcal{T}_i$ of $\mathcal{R}$ that can be assigned priority level i (line 5). Since the quantum can take $\|\psi\| = \psi_{\max} - \psi_{\min} + 1$ different values, there are $\|\psi\|^{\#\mathcal{T}_i}$ different quantum assignments for each subset $\mathcal{T}_i$. The basic exhaustive tree-search used to set priorities is the same as in the *Audsley-RR-FPP* algorithm, see section 3. In the following, we call *priority-search-tree* the search tree modeling the priority choices. Here, we explain how we use a similar search to choose the quantum assignment for each possible set $\mathcal{T}_i$. A method that speeds-up the search by pruning away sub-trees that cannot contain a solution is provided in §4.2.

Similarly to *Audsley-RR-FPP*, the search for the schedulable subset $\mathcal{T}_i$ is performed by exploring the priority-search-tree according to a depth-first strategy. When a leaf is reached, the schedulability of $\mathcal{T}_i$ is assessed. If $\mathcal{T}_i$ is feasible, the algorithm returns, otherwise, it backtracks till the first vertex such that not all its child vertices have been explored. To assess the schedulability of $\mathcal{T}_i$, all possible quantum assignments are successively considered. In the same manner as for the priority allocation, a tree -called *quantum-search-tree*- reflects the choices for quantum values. A depth-first strategy is used as well to explore this search space. In this case, a node has $\|\psi\|$ children where each child models a different quantum value. Here, we label the edge between vertices of depth k and $k+1$ with the quantum value of the $(k+1)^{th}$ task of $\mathcal{T}_i$. Thus, a vertex of depth k models the choices performed for the k first tasks of $\mathcal{T}_i$.

4.2 Complexity and improvements

Size of the search space. Assigning n tasks to different non-empty layers is like subdividing a set of n elements into non-empty subsets. Let k be the number of layers. The number of possible assignments is equal, by definition, to the the Stirling number of the second kind (see [1], page 824):

$$\frac{1}{k!}\sum_{i=0}^{k}(-1)^{(k-i)}\left(\begin{array}{c} k \\ i \end{array}\right) i^n,$$

where $\left(\begin{array}{c} k \\ i \end{array}\right)$ is the binomial coefficient, i.e., the number of ways of picking an unordered subset of i elements in a set of k elements.

The complexity depends on the number of tasks scheduled under RR since their quantum values have to be chosen. When there are k layers, at least $n-k+1$ tasks are in an RR layer (i.e., $n-k+1$ tasks in a single RR layer and one task in each of the remaining $k-1$ FPP layers) and up to $\max(n, 2(n-k))$ (i.e., tasks are "evenly" distributed among RR layers). Since the quantum can take $\|\psi\| = \psi_{\max} - \psi_{\min} + 1$ different values, there are between $\|\psi\|^{n-k+1}$ and $\|\psi\|^{\max(n,2(n-k))}$ different quantum assignments for a configuration of k layers.

In addition, n tasks can be subdivided into $k = 1, 2, ..., n$ many layers and there are $k!$ different possible priority orderings among the k priority layers. Thus, a lower bound for the search space of the problem of assigning priority, policy and quantum for a set of n tasks is

$$\sum_{k=1}^{n}\sum_{i=0}^{k}(-1)^{(k-i)} \cdot \left(\begin{array}{c} k \\ i \end{array}\right) \cdot i^n \cdot \|\psi\|^{n-k+1}.$$

In a similar way, we derive an upper bound by replacing $\|\psi\|^{n-k+1}$ with $\|\psi\|^{\max(n,2(n-k))}$.

For instance, as can be seen on figure 8, the size of the search space comprises about $2 \cdot 10^{11}$ scheduling configurations for a set of 10 tasks. The search space grows more than exponentially, thus an exhaustive search is not possible in practice in a wide range of real-time problems.

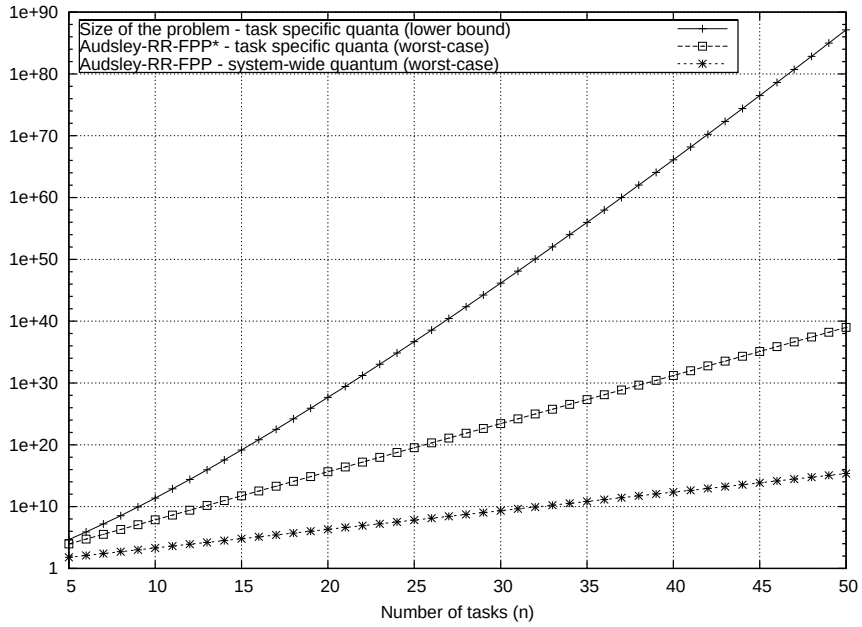


Figure 8: Complexity of the problem for a number of tasks varying from 5 to 50 (log. scale for y-axis). In the case of task-specific quanta, the quantum value ranges can be chosen in the interval $[1, 5]$.

***Audsley-RR-FPP**.** Our algorithm looks at each priority level i for a subset $\mathcal{T}_i$ in $\mathcal{R}$ which is schedulable at priority i (line 5). Since at least one task is assigned to each priority level, the number of tasks belonging to $\mathcal{R}$ when dealing with priority level i is lower than or equal to i. In addition, we know that there are $\|\psi\|^k$ different quantum assignments for a subset of k tasks. Thus, at each priority level i, the algorithm examines $\sum_{j=1}^{i} \left(\begin{array}{c} i \\ j \end{array} \right) \cdot \|\psi\|^j = (\|\psi\| + 1)^i - 1$ assignments in the worst-case. Thus, for priority level from 1 to n, the algorithm considers in the worst-case a number of assignments given by:

$$\sum_{i=1}^{n} (\|\psi\| + 1)^i - 1 = \frac{1 - (\|\psi\| + 1)^{n+1}}{1 - (\|\psi\| + 1)} - (n + 1)$$

This complexity for a varying number of tasks is shown on figure 8, for instance, for a set of 10 tasks with $\psi_{min} = 1$ and $\psi_{max} = 5$ it is approximately equal to $72 \cdot 10^6$. Figure 8 shows also the size of the search space and, for comparison, the worst-case complexity of *Audsley-RR-FPP*, *i.e.*, the case where the quantum size is a system-wide constant. Although we achieve a great complexity reduction with regards to an exhaustive search, the complexity remains exponential in the number of tasks, as previously noticed in section 3.

Complexity reduction. The idea is the same as for *Audsley-RR-FPP*. We want to reduce the number of sets that are to be considered. Property 3 given in this paragraph shows that it is possible to identify priority and policy assignments that are not schedulable whatever the quantum allocation. Thanks to property 2 and property 3, one can identify and prune away branches of the priority-search-tree which necessarily lead to subsets $\mathcal{T}_i$ that are not schedulable whatever the quantum assignments. Furthermore, with property 3, one can reduce in a similar manner the number of quantum assignments to consider for a particular subset $\mathcal{T}_i$ in a quantum-search-tree.

With the basic algorithm explains in §4.1, feasibility of a priority allocation is assessed at the leafs when all tasks have been given a priority by testing all quantum assignments. The idea is here to evaluate feasibility at intermediate vertices as well, by assigning a priority lower than i to the tasks for which no priority choice has been made yet. Under that configuration, if a task τ_i which is assigned the priority i is not schedulable whatever

the quantum assignment, there is no need to consider the children of this vertex. Indeed, from property 2, since the priority assignment of the children of this node will increase the set of same or higher priority tasks, the response time of τ_i cannot decrease. Thus, all child vertices corresponds to priority assignments that are not schedulable. Now, it remains to identify priority and policy assignments that are not schedulable whatever the quantum allocation. The following property, proven in appendix A.3, can be stated.

Property 3 *Let $\mathcal{S}$ be a schedulability test for which property 1 holds. Let $\mathcal{T}$ be a task set and $\mathcal{P}$ be a global priority and policy assignment. Let τ_i be a task with the maximum quantum value ψ_{max} in an RR layer. Let the quantum values of all other tasks in the RR layer be set to the minimum ψ_{min}. If the response time bound of τ_i, computed with $\mathcal{S}$, is greater than its relative deadline, then, whatever the quantum assignment under $\mathcal{P}$, τ_i will remain unschedulable with $\mathcal{S}$.*

Thus, at each vertex of the priority search tree, a priority assignment $\mathcal{P}$ is not feasible whatever the quantum assignment, if a task τ_k which has a priority i is not feasible with the quantum allocation given in property 3.

Similarly, we can cut branches when exploring the quantum-search-tree of a set $\mathcal{T}_i$. The idea is again to evaluate feasibility at intermediate vertices. Since an intermediate vertex models a partial quantum assignment for a set $\mathcal{T}_i$, we assign the lowest quantum value to each task in $\mathcal{T}_i$ which has no quantum assigned yet. In that case, if a task τ_k for which the quantum has already been set at this vertex is not schedulable, then there is no need to consider the children of this vertex. Indeed, given property 1, the response time of τ_k can only increase when the children of this vertex are considered.

The finding of this paragraph allows a very significant decrease in the average number of configurations tested by the *Audsley-RR-FPP** algorithm. For instance, for task sets constituted of 10 tasks, the algorithm examines on average only about 4000 configurations before coming up with a feasible solution or concluding that the task set is unfeasible while it would require about $7 \cdot 10^7$ tests otherwise. On the contrary to the system-wide quantum case, we are unable here to compare the *Audsley-RR-FPP** algorithm with and without the complexity reduction method since running the *Audsley-RR-FPP** without complexity reduction is too computationally intensive for task sets of cardinality higher than 10.

5 Heuristic approach for larger task sets

The exponential complexity of *Audsley-RR-FPP* and *Audsley-RR-FPP** is due to the number of different possible subsets $\mathcal{T}_i$ that can be assigned at each priority level i, see §3.2. Here, we present an heuristic, in the case of a system-wide quantum, that considers a limited number of $\mathcal{T}_i$ at each priority level i. The idea is to place the most CPU demanding tasks as soon as possible, and thus at the lowest priority levels. This algorithm, called *Load-RR-FPP,* is of polynomial complexity and thus better suited for larger task sets. The algorithm described below runs in $O(n^3)$ and thus is well suited up to 100 tasks. However, using the same scheme, it is possible to further reduce the search space to handle larger task sets.

5.1 *Load-RR-FPP heuristic*

A very similar algorithm to the one in section 3 is employed to find a schedulable configuration. The only difference lies in the search of a schedulable subset $\mathcal{T}_i$ at each priority level i (line 5 of algorithm 1). The heuristic tries to place the tasks with the highest utilisation rate at the considered priority level i. Sharing the CPU under RR enables large tasks to be feasible while being scheduled at low priority levels, and thus, it reduces the interference caused to the other tasks.

Let $\mathcal{L}$ be the list of tasks of $\mathcal{R}$ (*i.e.* tasks not yet assigned a priority and a policy) ordered by decreasing utilisation rate (C_k/T_k). Let $\mathcal{L}_k$ be the k^{th} task of $\mathcal{L}$, *i.e.*, the task of $\mathcal{R}$ with the k^{th} highest utilisation rate.

1. (Create $\mathcal{L}$, *i.e.*, sort the tasks of $\mathcal{R}$ by decreasing utilisation rate.)

2. (Look for a schedulable set $\mathcal{T}_i$.) Place in $\mathcal{T}_i$ the task with the highest utilisation rate, *i.e.*, $\mathcal{L}_1$. Then iteratively add the task with the next highest utilisation rate, *i.e.*, $\mathcal{L}_2$,$\mathcal{L}_3$, $\mathcal{L}_4$, etc...
   For j from 1 to $\#\mathcal{L}$,
   (a) (Put the j first tasks of $\mathcal{L}$ in $\mathcal{T}_i$.) $\mathcal{T}_i = \{\mathcal{L}_1, ..., \mathcal{L}_j\}$,
   (b) (Test feasibility.)
      i. ($\mathcal{T}_i$ feasible, return.) Return the set $\mathcal{T}_i$ feasible at priority i.
      ii. ($\mathcal{T}_i$ not feasible, continue.) Otherwise continue.
3. (Return failed.) No feasible set $\mathcal{T}_i$ was found, return $\emptyset$.

This heuristic can be further improved: when the set $\mathcal{T}_i$ is not feasible at step 2(b)ii, we iteratively remove from $\mathcal{T}_i$ some tasks that are not feasible. The step 2(b)ii becomes:

1. ($\mathcal{T}_i$ not feasible, remove unfeasible tasks.) Otherwise remove iteratively from $\mathcal{T}_i$ the unfeasible tasks. We remove first the tasks with the greatest utilisation rate.
   As long as there are unfeasible tasks in $\mathcal{T}_i$,
   (a) (Remove the unfeasible task with the highest utilisation rate.)
   (b) (Test feasibility.)
      i. ($\mathcal{T}_i$ feasible, return.) Return the set $\mathcal{T}_i$ feasible at priority i.
      ii. ($\mathcal{T}_i$ not feasible, continue.) Otherwise continue.

As it will be seen in the experiments of section 6, this heuristic is efficient and successfully finds a feasible set $\mathcal{T}_i$ in most cases.

5.2 Worst-case complexity

At each priority level i, the algorithm looks for a schedulable subset of tasks $\mathcal{T}_i$ in the set $\mathcal{R}_i$ (line 5 of algorithm 1). $\mathcal{R}_i$ is the set of task $\mathcal{R}$ with no assignment when the priority level i is considered. The complexity is thus equal to:

$$\sum_{i=1}^{n} \text{comp\_}\mathcal{R}_i \ , \tag{7}$$

where comp_$\mathcal{R}_i$ is the worst-case number of sets tested for a given $\mathcal{R}_i$.

Since at least one task is assigned to each priority level, the number of tasks belonging to $\mathcal{R}_i$ is lower than or equal to i, *i.e.*, $\#\mathcal{R}_i \leq i$. Since for a given set $\mathcal{R}_i$ the heuristic iteratively adds the tasks with the highest utilisation rate in $\mathcal{R}_i$ (step 2), there are initially $\#\mathcal{R}_i$ different sets tested. However, when the considered set $\mathcal{T}_i$ is not feasible, the unfeasible tasks are removed one by one (step 2(b)ii improved). At worst, all tasks are not feasible and thus $\#\mathcal{T}_i$ different sets $\mathcal{T}_i$ are considered each time the heuristic adds a task. So, the number of sets $\mathcal{T}_i$ considered for a given set $\mathcal{R}_i$ is:

$$\text{comp\_}\mathcal{R}_i = \sum_{j=1}^{\#\mathcal{R}_i} j = \frac{\#\mathcal{R}_i \cdot (\#\mathcal{R}_i + 1)}{2}, \tag{8}$$

thus from equations 7 and 8, the algorithm looks for:

$$\sum_{i=1}^{n} \sum_{j=1}^{i} j = \sum_{i=1}^{n} \frac{i \cdot (i+1)}{2} \Rightarrow \mathcal{O}(n^3) \ ,$$

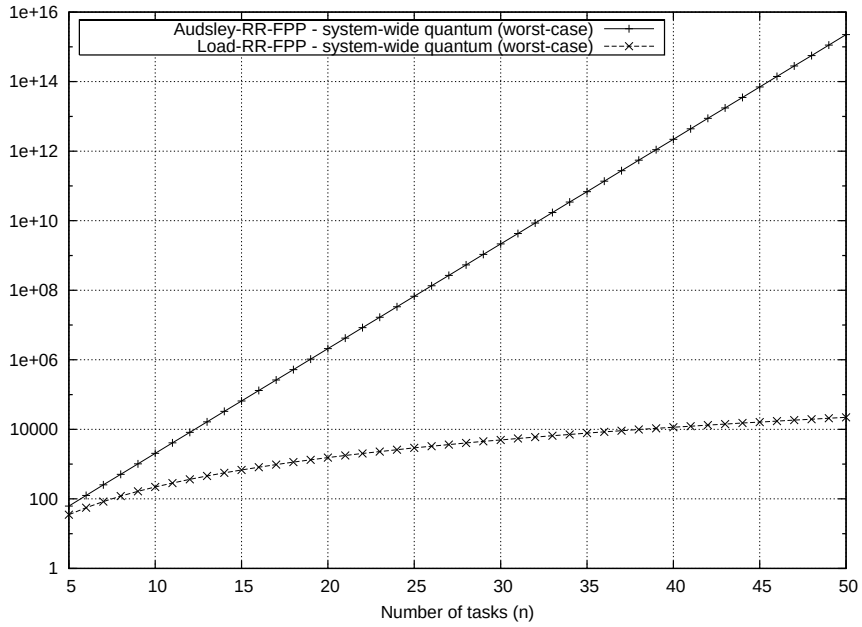


Figure 9: Number of configurations tested in the worst-case with *Audsley-RR-FPP* and *Load-RR-FPP* for a number of tasks varying from 5 to 50 when the quantum is a system-wide value (log. scale for y-axis).

assignments in the worst-case.

The complexity for a varying number of tasks is shown on figure 9. For instance, it is equal to 220 for a set of 10 tasks. For reference, figure 9 shows the complexity of *Audsley-RR-FPP*. As expected, one observes a great complexity reduction with regards to *Audsley-RR-FPP* for task sets of cardinality higher than 10.

6 Experimental results

This section first investigates the average complexity of the proposed algorithms: *Audsley-RR-FPP*, *Audsley-RR-FPP** and *Load-RR-FPP* in order to estimate the largest task sets that can be handled in practice. Then, the extent to which using both RR and FPP enables us to improve the schedulability is quantified by comparison with FPP alone. Here, we quantify the improvement brought when system-wide quantum or task-specific quanta are enabled by comparison with FPP. Furthermore, the efficiency of the heuristic *Load-RR-FPP* is assessed.

6.1 Experimental setup.

In the following experiments, we only consider task sets that are unschedulable with FPP alone. Since we choose to consider periodic tasks with deadlines equal to periods ($D_i = T_i$), we use the Rate Monotonic (FPP-RM) priority assignment, which is optimal in that context. The global load U (i.e., $\sum_{i=1}^{n} \frac{C_i}{T_i}$) has to be necessarily greater than $n \cdot (2^{1/n} - 1)$ (from [8]) in order to be able to exhibit non-feasible task sets. In the following, we choose a quantum value of 1 for the system-wide quantum[4] or, when task-specific quanta is considered, a quantum value which can be chosen in the interval $[1, 5]$. The parameters of an experiment are defined by the tuple (n, U) where n is the number of tasks and U is the global load. The utilization rate ($\frac{C_i}{T_i}$) of each task τ_i is uniformly distributed in the interval $\left[\frac{U}{n} \cdot 0.85, \frac{U}{n} \cdot 1.15\right]$. The computation time C_i is randomly chosen with an uniform law in the interval $[1, 50]$ and the period T_i is upper bounded by 1000. The results have been obtained with 200 task sets randomly generated with the aforementioned parameters.

[4]Experiments not detailed here suggest to us that on average the smaller the quantum value, the easier it is to find a schedulable priority and policy assignment. Intuitively, this is because usually smaller quanta allow to diminish the interference caused by same-priority tasks during a given time span.

6.2 Complexity of the computation

In this paragraph, we evaluate the number of trials needed to come up with a solution, *i.e.* providing a feasible assignment or concluding that the task set is not feasible. Task sets characterized by the tuple $(10, U)$ are randomly generated with a global load U ranging from 0.75 to 0.95. For each load of the task set, the average complexity for 1) *Audsley-RR-FPP**(optimal solution, task-specific quanta), 2) *Audsley-RR-FPP* (optimal solution, system-wide quantum) and 3) *Load-RR-FPP* (heuristic, system-wide quantum) computed over 200 task sets is shown on figure 10.

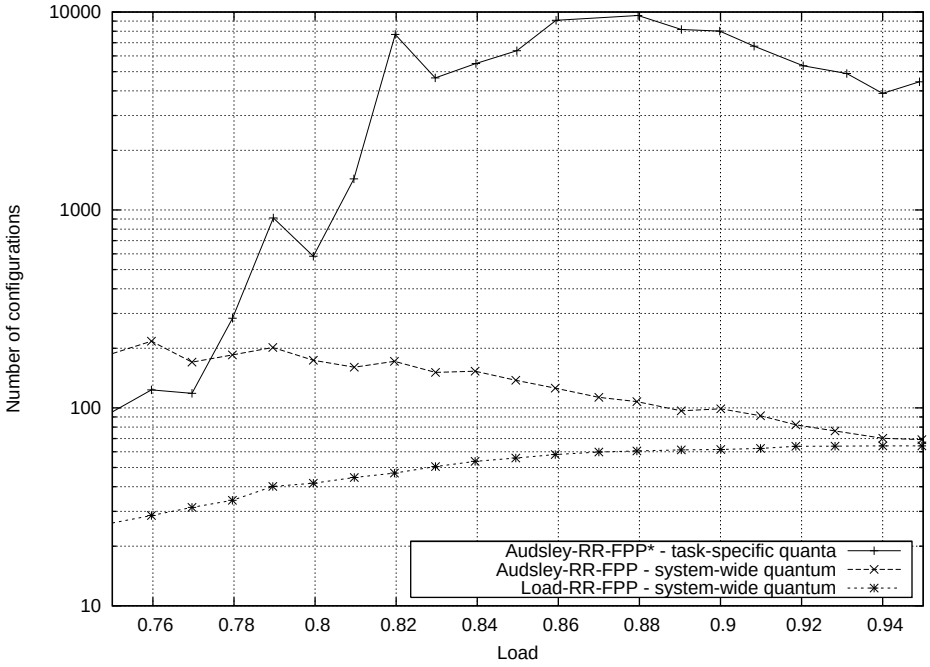


Figure 10: Average number of configurations tested by 1) *Audsley-RR-FPP**(task-specific quanta), 2) *Audsley-RR-FPP* (system-wide quantum) and 3) *Load-RR-FPP* (system-wide quantum). The CPU load ranges from 0.75 to 0.95. The number of tasks is equal to 10. Each point is the average value over 200 experiments. Log. scale is used for y-axis.

The methods discussed in §3.2 (complexity reduction in the case of a system-wide quantum) and §4.2 (complexity reduction in the case of task-specific quanta) are implemented to speed up the process. For instance, for a load of 0.88, the average number of configurations that need to be tested is about

1. 100 times smaller than the corresponding worst-case (*i.e.*, $72 \cdot 10^6$) with *Audsley-RR-FPP**,
2. 10 times smaller with *Audsley-RR-FPP* (*i.e.*, 2046).

We are unable to compare the results shown in figure 10 with the ones that would have been obtained without the complexity reduction strategies because in the latter case the computation time turns out to be prohibitive with task-specific quanta.

It is worth pointing out that, with task-specific quanta, the number of task sets tested increases up to a load of 86% and then decreases (see figure 10). This can be explained by two phenomena. On the one hand, it is easy to find quickly a schedulable configuration when the load is low. On the other hand, when the load increases, the technique described in §4.2 for cutting the search space is more efficient because unfeasible subtrees of the search tree are more easily identified. One also observes that the curve is not smooth, this is due to the high variability in the number of configurations tested[5] and could be improved with larger samples.

[5]The sampling distribution of the number of configurations tested has a standard deviation larger than 1000 and an excess kurtosis of 22.

The decreasing trend in the number of tested cases is even more acute with a system-wide quantum (*Audsley-RR-FPP*), indeed the pruning of the search space is more efficient because the quantum size being set reduces the possibilities of finding a feasible schedule. The heuristic which does not include a pruning procedure gets closer to its upper bound complexity $O(n^3)$ when the load increases because the complexity directly depends on the number of unfeasible tasks at each step. The heuristic becomes useful for larger task sets that cannot be handled by the optimal algorithm.

The limit of *Audsley-RR-FPP* and *Audsley-RR-FPP** is that, even with the pruning procedures, the complexity increases exponentially in the number of tasks. Further experiments show that task sets of cardinality 30 to 35 can be handled in a few hours with *Audsley-RR-FPP*. However, *Audsley-RR-FPP** is limited to smaller task sets (up to 20 tasks). Larger task sets can be treated with the heuristic *Load-RR-FPP* at the expense that results are no more optimal, see section 5 and paragraph 6.3 for experimental results.

6.3 Schedulability improvement over FPP

The same issue has been investigated in [10] using heuristics for assigning priorities and policies. The authors considered task sets with a load between 0.75 and 0.90 and found that more than 15% of the task sets unfeasible with FPP alone are feasible when using both FPP and RR. Here, we study the extent to which this can be improved using *Audsley-RR-FPP** and *Audsley-RR-FPP*, both optimal in their context, which provides more fine-grained elements to draw conclusions on the usefulness of RR for improving schedulability.

Figure 11 shows the percentage of task sets that are not schedulable with FPP alone and become schedulable when using 1) *Audsley-RR-FPP** (optimal solution, task-specific quanta), 2) *Audsley-RR-FPP* (optimal solution, system-wide quantum) and 3) *Load-RR-FPP* (heuristic, system-wide quantum) computed over 200 task sets.

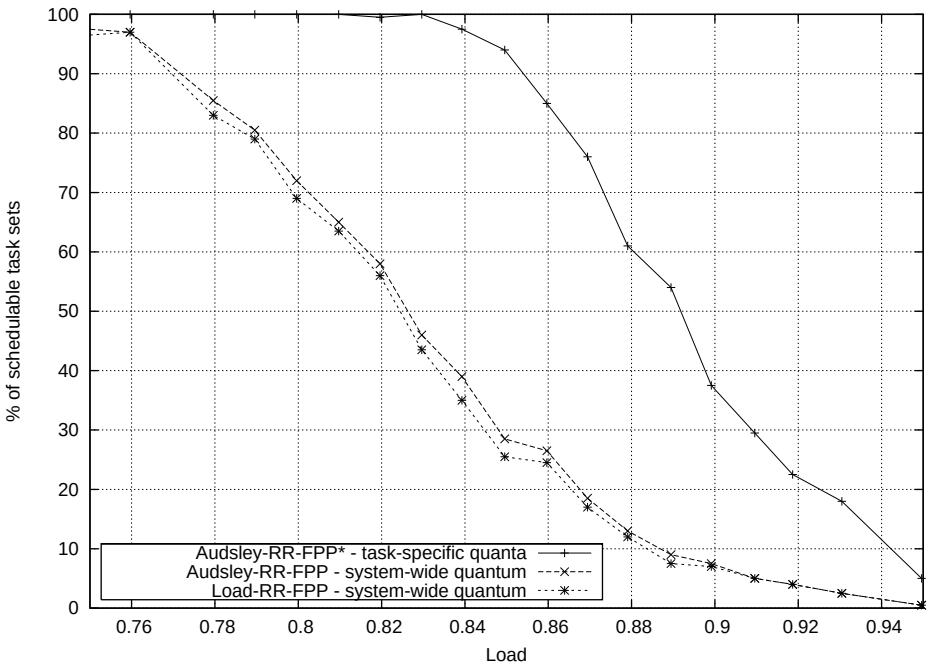


Figure 11: Percentage of task sets unschedulable with FPP-RM which become schedulable under Posix using 1) *Audsley-RR-FPP**(task-specific quanta), 2) *Audsley-RR-FPP* (system-wide quantum) and 3) *Load-RR-FPP* (system-wide quantum). The CPU load ranges from 0.75 to 0.95 and the number of tasks is equal to 10. Each point is the average value over 200 experiments.

Task sets are randomly generated according to the tuple $(10, U)$ where the global load U varies from 0.75 to 0.95. One observes that using both RR and FPP, a large number of task sets not schedulable with FPP alone

becomes schedulable: it ranges from nearly 100% for a load equal to 0.75 to about 4% for a load of 0.95 with *Audsley-RR-FPP**. The downward trend was expected given that, in general, highly loaded task sets are more difficult to schedule successfully. Anyway, this experiment gives clear-cut evidence that using both FPP and RR is beneficial for schedulability, even at relatively high workload.

One observes that the improvement with task-specific quanta is very important, at least 3 times better than with a system-wide quantum when the load is higher than 85%. When the load is lower than 83%, a solution is found in almost all cases with task-specific quanta, the percentage of successes remaining greater than 50% up to a load equal to 89%. As it was expected, when the load gets higher, feasible scheduling solution tends to rarefy. One can observe the same trend for system-wide quantum: when the load is lower than 79%, a solution is found in at least 80% of cases. The percentage of successes remaining greater than 40% up to a load equal to 83%. Figure 11 shows also that the *Load-RR-FPP* heuristic is not far from the optimal solution provided by *Audsley-RR-FPP* and allows most often to find a schedulable configuration when one exists.

6.4 Influence of the size of the task sets

Experiments not shown here suggest that the number of schedulable task sets slightly decreases when the number of tasks increases. This result is quite surprising because intuitively, the higher the number of tasks, the smaller they are and the higher is the degree of freedom to spread the workload over different priority levels. This counter-intuitive result is probably due to the pessimism of the schedulability analysis used in this study. Indeed, our experiments show that on average the response time bound of at least 90% of the tasks in a RR layer is computed with the first term $\left\lceil \frac{s_{i,j}}{\psi_i} \right\rceil \cdot (\overline{\psi}_i - \psi_i) + \widetilde{s}_i(t)$ of equation 5. This value depends on the quantum values of the other tasks in the RR layer and not on the actual workload brought by the other tasks in the RR layer. The bound is thus often pessimistic and it is worst when the number of tasks in the RR layer gets high. To some extent, it seems that this phenomenon evens out the benefits of the tasks being smaller when the number of tasks is high. This may explain why the number of tasks slightly decreased the percentage of schedulable task sets in our experiments.

The same trend can be observed with *Load-RR-FPP* heuristic: figure 12 presents the percentage of task sets unschedulable with FPP-RM for which *Load-RR-FPP* heuristic finds a schedulable configuration. The task sets are randomly generated according to $(n, 0.84)$ where the number of tasks n varies from 25 to 45. This experiments highlights that the heuristic is usable with much larger task sets than with *Audsley-RR-FPP** and *Audsley-RR-FPP*.

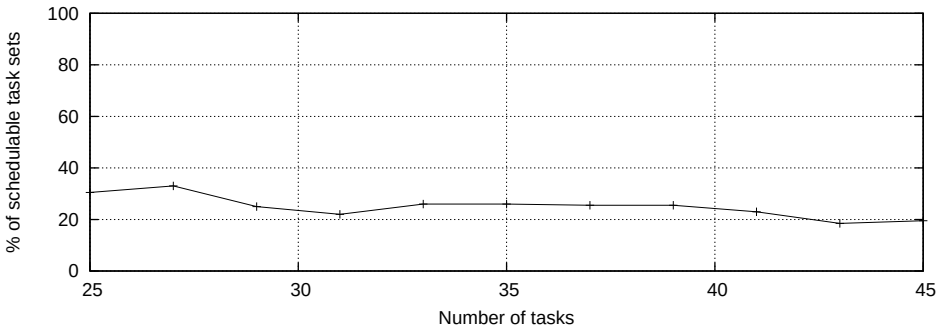


Figure 12: Percentage of task sets unschedulable with FPP-RM for which the heuristic *Load-RR-FPP* finds a schedulable configuration with the combined used of RR (system-wide quantum) and FPP. The CPU load is equal to 0.84 and the number of tasks ranges from 25 to 45.

Our experiments show that the combined used of RR and FPP with process-specific quanta allows to schedule a large number of task sets which are neither schedulable with FPP nor with a system-wide quantum. It is

worth noting that context switch latencies were neglected while RR induces more context switches than FPP. This fact weakens to a certain extent our conclusions. A future work is to find the feasible quantum allocation that minimizes the global number of context switches.

7 Conclusion

In this technical report, we propose algorithms for assigning priority, policy and quantum on Posix 1003.1b compliant OS, distinguishing the case where the quantum is a system-wide constant and the case where it can chosen on a task-per-task basis. It has been shown that these algorithms are optimal in the sense that if there is at least one schedule, that can be identified as feasible by the schedulability test, a feasible solution will be returned by the algorithms.

In terms of worst-case complexity, the algorithms greatly improve upon a brute force approach but are still exponential in the number of tasks. Therefore, they are not suited to large task sets. That is why we propose the *Load-RR-FPP* heuristic that is able to handle efficiently larger systems. A future work is to take into account context switches and come up with a way of assigning quantum values in such a way as minimizing the context-switch overhead.

A result yields by the experiments is that the combined used of FPP and RR with task-specific quanta enables to significantly improve schedulability by comparison with FPP alone and with system-wide quantum. This is particularly interesting in the context of embedded systems where the cost pressure is high, and thus lead us to exploit the computational resources at their fullest.

A Proof of properties 1 and 2

In this appendix, we prove that the schedulability analysis [9] ensures that properties 1 and 3 hold. The first paragraph is devoted to the study of the execution end $e_{i,j}$ of $\tau_{i,j}$ computed with [9] under two configurations $(\mathcal{P}, \Psi_{\mathcal{P}})$ and $(\mathcal{P}, \Psi'_{\mathcal{P}})$ that only differ by their quantum assignment. This result is used in subsequent proofs.

A.1 Execution end bound: basic properties

We compare bounds on the execution end of τ_i under the same priority and policy assignment $\mathcal{P}$ with two different quantum allocations. Let $e_{i,j}$ and $e'_{i,j}$ be respectively the execution end bound of τ_i under $(\mathcal{P}, \Psi_{\mathcal{P}})$ and under $(\mathcal{P}, \Psi'_{\mathcal{P}})$. Since τ_i is in an RR layer, $e_{i,j}$ is computed with equation 4 of §2.4:

$$e_{i,j} = \min\{t > 0 \mid \overline{\Psi}_i(t) + s_{i,j} = t\} \ ,$$

where (equation 5 of §2.4)

$$\overline{\Psi}_i^{\Psi_{\mathcal{P}}}(t) = \min\left(\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) + \widetilde{s}_i(t), s_i^*(x)\right) \ ,$$

where $\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}$ is the sum of the quanta of all other tasks of the RR layer. Since $s_{i,j}$, $\widetilde{s}_i(t)$ and $s_i^*(x)$ are independent of the quantum assignment (see §2.4), it is enough to compare the first term of the min() to decide which task will have the smallest response time bound. Two cases arise:

1. $\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) > \left\lceil \frac{s_{i,j}}{\psi_i^{\Psi'_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi'_{\mathcal{P}}} - \psi_i^{\Psi'_{\mathcal{P}}})$ then we conclude $e_{i,j} \geq e'_{i,j}$,
2. otherwise:

$$\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) \leq \left\lceil \frac{s_{i,j}}{\psi_i^{\Psi'_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi'_{\mathcal{P}}} - \psi_i^{\Psi'_{\mathcal{P}}}) \ ,$$

and $e_{i,j} \leq e^{'}_{i,j}$.

When $s^*_i(x)$ is the minimum, we have $e_{i,j} = e^{'}_{i,j}$.

From this finding we can deduce that for any other assignment $\Psi^{'}_{\mathcal{P}}$, if the two following requirements are met:

requirement 1: the quantum $\psi_i^{\Psi^{'}_{\mathcal{P}}}$ of τ_i in $\Psi^{'}_{\mathcal{P}}$ is lower than or equal to its quantum $\psi_i^{\Psi_{\mathcal{P}}}$ under $\Psi_{\mathcal{P}}$,

requirement 2: the sum of the quanta of all other tasks of the RR layer $\mathcal{T}_i^{\mathcal{P}}$ under $\Psi^{'}_{\mathcal{P}}$ is greater than or equal to the one under $\Psi_{\mathcal{P}}$, i.e., $\overline{\psi}_i^{\Psi^{'}_{\mathcal{P}}} - \psi_i^{\Psi^{'}_{\mathcal{P}}} \geq \overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}$ where $\overline{\psi}_i^{\Psi_{\mathcal{P}}} = \sum_{\tau_k \in \mathcal{T}_i} \psi_{\tau_k}^{\Psi_{\mathcal{P}}}$ is the sum of the quantum of all tasks of the RR layer $\mathcal{T}_i^{\mathcal{P}}$ under quantum allocation $\Psi_{\mathcal{P}}$,

then we have:

$$\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi^{'}_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi^{'}_{\mathcal{P}}} - \psi_i^{\Psi^{'}_{\mathcal{P}}}) \geq \left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) \ , \tag{9}$$

and thus $\forall \tau_{i,j}$, $e_{i,j} \leq e^{'}_{i,j}$ which implies that the response time bound of τ_i under $(\mathcal{P}, \Psi^{'}_{\mathcal{P}})$ is greater than or equal to the response time bound under $(\mathcal{P}, \Psi_{\mathcal{P}})$.

A.2 Proof of property 1

Since the prerequisites of property 3 are exactly requirements 1 and 2 of §A.1, the response time bound of τ_i in property 3, is no less under $(\mathcal{P}, \Psi^{'}_{\mathcal{P}})$ than under $(\mathcal{P}, \Psi_{\mathcal{P}})$. Since τ_i is not schedulable under $(\mathcal{P}, \Psi_{\mathcal{P}})$, it cannot be schedulable under $(\mathcal{P}, \Psi^{'}_{\mathcal{P}})$.

A.3 Proof of property 3

We show that the bound on the execution end $e_{i,j}$ for a task in an RR layer under $\mathcal{P}$, is minimum under $\mathcal{P}$ when the quantum of τ_i is equal to ψ_{max} while the quanta of the other tasks in the layer are set to ψ_{min}. Let $\Psi_{\mathcal{P}}$ be the corresponding quantum assignment where

$$\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) = \left\lceil \frac{s_{i,j}}{\psi_{max}} \right\rceil \cdot \Big(\sum_{\tau_k \in \mathcal{T}_{p_i}^{\mathcal{P}} \setminus \{\tau_i\}} \psi_{min} \Big)$$

and one notes that whatever a different quantum assignment $\Psi^{'}_{\mathcal{P}}$:

$$\left\lceil \frac{s_{i,j}}{\psi_{max}} \right\rceil \cdot \sum_{\tau_k \in \mathcal{T}_{p_i}^{\mathcal{P}} \setminus \{\tau_i\}} \psi_{min} \leq \left\lceil \frac{s_{i,j}}{\psi_i^{\Psi^{'}_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi^{'}_{\mathcal{P}}} - \psi_i^{\Psi^{'}_{\mathcal{P}}})$$

since, by definition, $\psi_{max} \geq \psi_i^{\Psi^{'}_{\mathcal{P}}}$ and $\psi_{min} \leq \psi_k^{\Psi^{'}_{\mathcal{P}}}$. From equation 9, the execution end bound $e_{i,j}$ of $\tau_{i,j}$ is thus minimum with $\Psi_{\mathcal{P}}$ among the set of all possible quantum assignments.

B Proof of property 2

Let $(\mathcal{P}, \Psi_{\mathcal{P}})$ and $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ be two scheduling configurations such as, for an arbitrary task τ_i:

1. the set of tasks with higher priority than τ_i under $\mathcal{Q}$, denoted by $\mathcal{T}^{\mathcal{Q}}_{hp(p_i)}$, is a subset of the set of tasks with higher priority than τ_i under $\mathcal{P}$, i.e., $\mathcal{T}^{\mathcal{Q}}_{hp(p_i)} \subseteq \mathcal{T}^{\mathcal{P}}_{hp(p_i)}$,
2. the set of tasks with equal priority than τ_i under $\mathcal{Q}$, noted $\mathcal{T}^{\mathcal{Q}}_{p_i}$, is a subset of the set of tasks with equal priority than τ_i under $\mathcal{P}$, i.e., $\mathcal{T}^{\mathcal{Q}}_{p_i} \subseteq \mathcal{T}^{\mathcal{P}}_{p_i}$,

3. and the tasks with equal priority than τ_i under $\mathcal{Q}$, *i.e.*, the tasks of $\mathcal{T}_{p_i}^{\mathcal{Q}}$, have the same quantum value under $\Psi_{\mathcal{Q}}$ than under $\Psi_{\mathcal{P}}$, *i.e.*, $\forall \tau_j \in \mathcal{T}_{p_i}^{\mathcal{Q}}$, $\psi_j^{\Psi_{\mathcal{Q}}} = \psi_j^{\Psi_{\mathcal{P}}}$.

This part shows that the response time bound under $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ is lower or equal than the response time bound under $(\mathcal{P}, \Psi_{\mathcal{P}})$ and thus property 2 holds.

Since the response time bound of τ_i is expressed with equation 2 of §2.4, then property 2 holds if and only if:

$$\begin{array}{c} j^* = \min\{j \mid (e_{i,j}^{\mathcal{Q}} \leq a_{i,j+1}^{\mathcal{Q}}) \vee (e_{i,j}^{\mathcal{P}} \leq a_{i,j+1}^{\mathcal{P}})\}, \\ \forall j \leq j^*,\ e_{i,j}^{\mathcal{Q}} - a_{i,j+1}^{\mathcal{Q}} \leq e_{i,j}^{\mathcal{P}} - a_{i,j+1}^{\mathcal{P}}\ , \end{array} \tag{10}$$

where $e_{i,j}^{\mathcal{P}}$ is the value of $e_{i,j}$ under $(\mathcal{P}, \Psi_{\mathcal{P}})$. Since the release time $a_{i,j}$ is independent of the assignment, then $a_{i,j}^{\mathcal{P}} = a_{i,j}^{\mathcal{Q}}$ forall i. Hence, equation 10 becomes:

$$\begin{array}{c} j^* = \min\{j \mid \min(e_{i,j}^{\mathcal{Q}}, e_{i,j}^{\mathcal{P}}) \leq a_{i,j+1})\}, \\ \forall j \leq j^*,\ e_{i,j}^{\mathcal{Q}} \leq e_{i,j}^{\mathcal{P}}\ . \end{array} \tag{11}$$

The computation of $e_{i,j}^{\mathcal{P}}$ under $(\mathcal{P}, \Psi_{\mathcal{P}})$ is performed with equation 4 (RR) or 3 (FPP). In these equations, only $\widetilde{s}_i^{\mathcal{P}}(t)$,$(\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}})$ and $\overline{s}_i^{\mathcal{P}}(x)$ depend on the scheduling configuration $(\mathcal{P}, \Psi_{\mathcal{P}})$. One notes that the functions $\widetilde{s}_i(t)$, $(\overline{\psi}_i - \psi_i)$ and $\overline{s}_i(x)$ under $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ are lower than or equal to these functions under $(\mathcal{P}, \Psi_{\mathcal{P}})$. Indeed, from requirement 1 and 2:

$$\begin{array}{c} \mathcal{T}_{hp(p_i)}^{\mathcal{Q}} \subseteq \mathcal{T}_{hp(p_i)}^{\mathcal{P}} \\ \\ \Longrightarrow \sum_{\tau_k \in \mathcal{T}_{hp(p_i)}^{\mathcal{Q}}} s_k(t) \leq \sum_{\tau_k \in \mathcal{T}_{hp(p_i)}^{\mathcal{P}}} s_k(t) \\ \Longrightarrow \widetilde{s}_i^{\mathcal{Q}}(t) \leq \widetilde{s}_i^{\mathcal{P}}(t) \end{array} \tag{12}$$

Requirement 2 and 3 are:

$$\begin{array}{c} \mathcal{T}_{p_i}^{\mathcal{Q}} \subseteq \mathcal{T}_{p_i}^{\mathcal{P}} \text{ and } \forall \tau_j \in \mathcal{T}_{p_i}^{\mathcal{Q}},\ \psi_j^{\Psi_{\mathcal{Q}}} = \psi_j^{\Psi_{\mathcal{P}}} \\ \Longrightarrow \overline{\psi}_i^{\Psi_{\mathcal{Q}}} - \psi_i^{\Psi_{\mathcal{Q}}} \leq \overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}} \end{array} \tag{13}$$

$$\text{and } \overline{s}_i^{\mathcal{Q}}(u+x) \leq \overline{s}_i^{\mathcal{P}}(u+x) \tag{14}$$

Since $e_{i,j}$ is computed with equation 3 (resp. equation 4) when τ_i is in a FPP (resp. RR) layer both under $\mathcal{P}$ and $\mathcal{Q}$, $e_{i,j}^{\mathcal{Q}} \leq e_{i,j}^{\mathcal{P}}$ from inequalities 12, 13, and 14. Thus, the equation 11 is true and the property 2 holds. Due to space limitation, the case where τ_i is in an RR layer under $\mathcal{P}$ ($e_{i,j}^{\mathcal{P}}$ computed with equation 4) while τ_i is in an FPP layer under $\mathcal{Q}$ ($e_{i,j}^{\mathcal{Q}}$ computed with equation 3) is not developed here but can be shown in the same manner.

C Proof of optimality

Here we show that the *Audsley-RR-FPP** algorithm is optimal in the sense that if there is a priority, policy and quantum assignment that can be identified as feasible by the schedulability analysis, it will be found by the algorithm[6]. Let us first remind the following theorem which has been proven in [2, 3, 5, 9] for various contexts of fixed priority scheduling.

[6]Since *Audsley-RR-FPP* is a special case of *Audsley-RR-FPP**where the quantum value is unique, the proof can be applied to *Audsley-RR-FPP*.

Theorem 1 *[3] Let $(\mathcal{P}, \Psi_{\mathcal{P}})$ be a schedulable configuration up to priority i, i.e. tasks that have been assigned the priorities from n to i are schedulable. If there exists a schedulable configuration $(\mathcal{A}, \Psi_{\mathcal{A}})$, then there is at least one schedulable configuration $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ having an identical configuration as $(\mathcal{P}, \Psi_{\mathcal{P}})$ for priorities n to i.*

From theorem 1, we can prove the optimality of *Audsley-RR-FPP**. Indeed, if *Audsley-RR-FPP** happens to fail at level i, the priority, scheduling policy and quantum assignment $(\mathcal{P}, \Psi_{\mathcal{P}})$ provided by *Audsley-RR-FPP** leads to a schedulable solution up to level $i+1$. Since *Audsley-RR-FPP** performs an exhaustive search to assign level i, there cannot be any *schedulable* assignment $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ possessing the same assignment as $(\mathcal{P}, \Psi_{\mathcal{P}})$ for priority $i+1$ to n. Thus, from theorem 1, there is no schedulable assignment.

We give here an intuitive proof of theorem 1, which basically is valid under Posix thanks to lemma 1 and property 2. It should be pointed out that theorem 1, and thus the optimality result of *Audsley-RR-FPP**, does not hold where property 2 is not verified by the schedulability test.

Theorem 1 holds if a schedulable configuration $(\mathcal{A}, \Psi_{\mathcal{A}})$ can be transformed into a schedulable configuration $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ for which the configuration is the same as $(\mathcal{P}, \Psi_{\mathcal{P}})$ for priority i to n. This transformation can be done iteratively by changing the configuration of certain tasks in $(\mathcal{A}, \Psi_{\mathcal{A}})$ to the configuration they have in $(\mathcal{P}, \Psi_{\mathcal{P}})$. The procedure is the following: for priority level k from n to i, assign in $(\mathcal{A}, \Psi_{\mathcal{A}})$ the priority $k+n-i$ to the tasks of priority k in $(\mathcal{P}, \Psi_{\mathcal{P}})$ (i.e., the set $\mathcal{T}_k^{\mathcal{P}}$) and set their quantum value to their values $\psi_i^{\mathcal{P}}$ in $\Psi_{\mathcal{P}}$ ($\forall \tau_j \in \mathcal{T}_k^{\mathcal{P}}$, $p_j^{\mathcal{A}} = p_j^{\mathcal{P}} + n - i$, $sched_j^{\mathcal{A}} = sched_j^{\mathcal{P}}$ and $\psi_j^{\Psi_{\mathcal{A}}} = \psi_j^{\Psi_{\mathcal{P}}}$). Since at each step, tasks in $\mathcal{T}_k^{\mathcal{P}}$ have the same quantum assignment, the same set of higher and equal priority tasks under the current configuration $(\mathcal{A}, \Psi_{\mathcal{A}})$ as under $(\mathcal{P}, \Psi_{\mathcal{P}})$, they remain schedulable under $(\mathcal{A}, \Psi_{\mathcal{A}})$ by lemma 1. From property 2, the other tasks $(\mathcal{T} \setminus \mathcal{T}_k^{\mathcal{P}})$ meet their deadline too since the quantum assignment and the set of higher and same priority task is reduced or stay unchanged under current configuration compared to the initial configuration $(\mathcal{A}, \Psi_{\mathcal{A}})$. Note that in the proof the priority range has been artificially extended by adding $n-i$ lower priority levels in order to avoid the case where a higher priority tasks is moved to a non-empty layer since property 2 does not cover this situation.

Notations

- $\mathcal{T} = \{\tau_1, ..., \tau_n\}$: a set of n periodic tasks
- $\mathcal{P}$: priority and policy assignment
- $\Psi_{\mathcal{P}}$: a specific quantum allocation under assignment $\mathcal{P}$
- $(\mathcal{P}, \Psi_{\mathcal{P}})$: a priority, policy and a quantum assignment
- $\mathcal{T}_i^{\mathcal{P}}$: subset of tasks assigned to priority level i under $\mathcal{P}$
- $\mathcal{T}_{hp(i)}^{\mathcal{P}}$: subset of tasks assigned to a higher priority than i under $\mathcal{P}$
- $\mathcal{T}_{lp(i)}^{\mathcal{P}}$: subset of tasks assigned to a lower priority than i under $\mathcal{P}$
- $\psi_i^{\Psi_{\mathcal{P}}}$: Round-Robin quantum for task τ_i under $\Psi_{\mathcal{P}}$
- $\overline{\psi}_i^{\Psi_{\mathcal{P}}}$: sum of the quanta of all tasks in layer $\mathcal{T}_i$ under $\Psi_{\mathcal{P}}$
- $s_i(t) = C_i \cdot \left\lceil \frac{t}{T_i} \right\rceil$: majorizing work arrival function on an interval of length t for a periodic task τ_i
- $\widetilde{s}_i(t) = \sum_{\tau_k \in \mathcal{T}_{hp(p_i)}^{\mathcal{P}}} s_k(t)$: the demand from higher priority tasks under $\mathcal{P}$
- $s_{i,j} = \sum_{i=1}^{j} C_i$: the demand from previous instances plus demand of current instance $\tau_{i,j}$ of τ_i
- $\overline{s}_i(x) = \sum_{\tau_k \in \mathcal{T}_{p_i}^{\mathcal{P}} \setminus \{\tau_i\}} s_k(x)$ is the demand from all other tasks than τ_i at priority level i under assignment $\mathcal{P}$.

References

[1] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions.* Dover Publications (ISBN 0-486-61272-4), 1970.

[2] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Report YO1 5DD, Dept. of Computer Science, University of York, England, 1991.

[3] N.C. Audsley. On priority assignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1):39–44, 2001.

[4] R. Brito and N. Navet. Low-power round-robin scheduling. In *Proc. of the 12th international conference on real-time systems (RTS 2004)*, 2004.

[5] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA, 1996. Available at `http://www.inria.fr/rrrt/rr-2966.html`.

[6] Mathieu Grenier and Nicolas Navet. Improvement in the configuration and analysis of Posix 1003.1b scheduling. In *Proc. of the 15th Real-Time and Network Systems (RTNS'07)*, Nancy, France, 2007.

[7] (ISO/IEC) 9945-1:2004 and IEEE Std 1003.1, 2004 Edition. Information technology—portable operating system interface (POSIX®)—part 1: Base definitions. IEEE Standards Press, 2004.

[8] C.L. Liu and J.W Layland. Scheduling algorithms for multiprogramming in hard-real time environnement. *Journal of the ACM*, 20(1):40–61, 1973.

[9] J. Migge, A. Jean-Marie, and N. Navet. Timing analysis of compound scheduling policies : Application to Posix1003.1b. *Journal of Scheduling*, 6(5):457–482, 2003.

[10] N. Navet and J. Migge. Fine tuning the scheduling of tasks through a genetic algorithm: Application to Posix1003.1b compliant OS. *Proc. of IEEE Proceedings Software*, 150(1):13–24, 2003.

[11] K. Tindell. An extendible approach for analyzing fixed priority hard real-time tasks. Technical Report YCS-92-189, Department of Computer Science, University of York, 1992.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399