Robust Partitioned Scheduling for Static-Priority Real-Time Multiprocessor Systems with Shared Resources

Frédéric Fauberteau, Serge Midonnet

To cite this version:

Frédéric Fauberteau, Serge Midonnet. Robust Partitioned Scheduling for Static-Priority Real-Time Multiprocessor Systems with Shared Resources. 18th International Conference on Real-Time and Network Systems, Nov 2010, Toulouse, France. pp.217-225. hal-00546968

HAL Id: hal-00546968
https://hal.science/hal-00546968

Submitted on 15 Dec 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Robust Partitioned Scheduling for Static-Priority Real-Time Multiprocessor Systems with Shared Resources

Frédéric Fauberteau
Université Paris-Est
LIGM, UMR CNRS 8049
5, bd Descartes, Champs-sur-Marne,
77454 Marne-la-Vallée CEDEX 2, France
Email: frederic.fauberteau@univ-paris-est.fr

Serge Midonnet
Université Paris-Est
LIGM, UMR CNRS 8049
5, bd Descartes, Champs-sur-Marne,
77454 Marne-la-Vallée CEDEX 2, France
Email: serge.midonnet@univ-paris-est.fr

Abstract

We focus on the partitioned scheduling of sporadic real-time tasks with constrained deadlines. The scheduling policy on each processor is static-priority. The considered tasks are not independent and the consistency of these shared data is ensured by a multiprocessor synchronization protocol. Considering these assumptions, we propose a partitioned scheduling algorithm which tends to maximize the robustness of the tasks to the Worst Case Execution Time (WCET) overruns faults. We describe the context of the problem and we outline our solution based on simulated annealing.

1 Introduction

Since the multiprocessor platforms have become predominant, the multiprocessor scheduling takes up an important place in the study of real-time scheduling. The literature concerning the multiprocessor scheduling contains many references about various classes of scheduling.

The uniprocessor scheduling algorithms are generally classified by type of priority mechanism. We distinguish three main classes of priority mechanism: *static-priority* (e.g. *Rate-Monotonic* (RM) [23], *Deadline-Monotonic* (DM) [22]), *task-level dynamic* (e.g. *Earliest-Deadline-First* (EDF) [23]) and *fully dynamic* (e.g. *Least-Laxity-First* (LLF) [24]). In this paper, we focus on a multiprocessor scheduling with a static-priority mechanism.

In addition to the criterion of complexity of the priority mechanism, the multiprocessor scheduling are also classified by a second criterion: the degree of migration allowed. We distinguish three main classes of migration degrees: *no migration* (i.e. partitioned scheduling), *restricted migration* and *full migration* (i.e. global scheduling).

In restricted migration, the migrations are allowed at jobs boundaries. Even if the restricted migration has been less studied than the two other approaches (partitioned and global), a restricted migration scheduling algorithm have been proposed by Baruah and Carpenter [3]. In global scheduling, the jobs are allowed to migrate. This approach has been more largely studied and several algorithms has been proposed for the three classes of priority mechanism. The RM and EDF approach for global scheduling has been introduced by [13]. More recently, the global algorithms P-Fair [4] and LLREF [11] has been proved to be optimal.

The global approach is very attractive but the cost of the migrations on some platforms can be difficult to estimate. We give for example the topology of one of our multi-core/multiprocessor platform in Figure 1. This platform is composed by 2 Intel® Xeon® E5410 at 2.33GHz. Each processor is a quad-core processor but we notice that for $0 \leq x \leq 3$, only the cores $P\#[x]$ and $P\#[x+4]$ share a cache of level 2. It can be interesting to think that a migration of a job from the core $P\#0$ to the core $P\#2$ may have a higher cost than a migration from the core $P\#0$ to the core $P\#4$. Considering we cannot precisely estimate the cost of a migration between any pair of cores, we focus on a partitioned scheduling in this paper.

The partitioned scheduling approach has also been well studied. Since no migrations are allowed, a partitioned scheduling of n tasks on m processors can

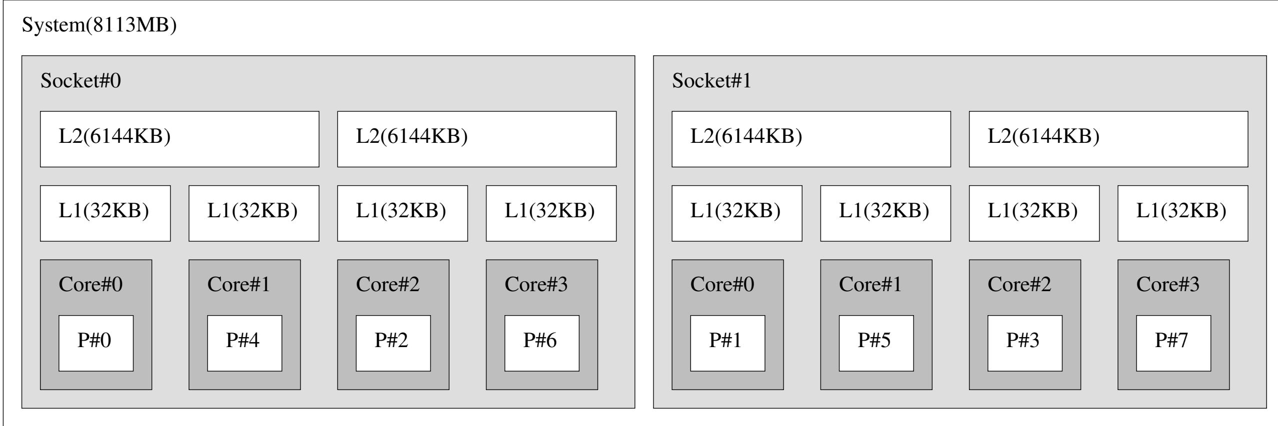


Figure 1. Topology of the multi-core/multiprocessor platform Dell Precision™ T7400.

be seen as m independent uniprocessor scheduling. The main challenge in the design of a partitioned scheduling is to find a feasible partition of a set of tasks. Since the partitioning can be seen as a instance of BIN-PACKING problem (which is NP-hard in the strong sense [18]), no optimal assignment can be found in polynomial time unless P=NP. Because the two problems are similar, the approximation methods used to solve instances of BIN-PACKING problem can be used to find a partition of a set of tasks. Therefore, many partitioned scheduling algorithms are built from a well-known approximation algorithm for the BIN-PACKING problem (e.g. FBB-FFD algorithm is built from *First-Fit* (FF) [16] and RM-DU-NFS algorithm is built from *Next-Fit* (NF) [1]).

The tasks considered in this paper are non-independent and several tasks can share the same resource. A multiprocessor synchronization protocol is needed to ensure the consistency of the shared data. This protocol shall avoid both deadlocks and unbounded priority inversions. The feasibility analysis to allocate a task to a processor no longer depends only on the tasks already allocated on this processor. The tasks allocated on other processors which share a resource with this task must be considered in the feasibility analysis.

The originality of this work is to allocate the tasks in order to improve the robustness to the WCET overruns faults. A task can commit a WCET overruns fault when its WCET has been underestimated or when this task suffers interference which leads to a greater execution requirement (*e.g.* not enough predictable OS). But techniques exist to compute the time Δt during which a task can continue to be executed without leading to a deadline miss. Our approach to improve the robustness is to maximize this time Δt for the all tasks of the system.

The rest of this paper is organized as follows. In Section 2, we introduce the terminology and we describe the context of this research. In Section 3, we discuss the proposed solution and we explain how to implement it. In Section 4, we show the results obtained from our implementation of the proposed solution. In Section 5, we conclude by presenting our perspective for this work.

2 Problem description

2.1 Terminology

We consider an application built from a set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n sporadic real-time tasks. A sporadic task is a recurring task for which only a lower bound on the separation between release times of the jobs is known. Each task τ_i is characterized by a minimum inter-arrival time T_i (also denoted period), a WCET C_i and a relative deadline D_i. The considered tasks are tasks with constrained deadlines (i.e. $D_i \leq T_i$). The utilization of the task τ_i is denoted u_i and is defined as $u_i = \frac{C_i}{T_i}$. This application runs on a platform $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ of m identical processors (homogeneous case). The total CPU utilization of a processor π_j is denoted $u_{sum}(\pi_j)$ and the utilization of a set of tasks τ_i is denoted $u_{sum}(\tau_i)$. We consider a static-priority scheduling on each processor. A static-priority scheduler assigns a priority to each task and all jobs of a task is released with the static priority of this task. The priorities of the tasks are taken in an increasing order (i.e. the priority of τ_{i+1} is higher than the priority of τ_i). The set of tasks with a priority lower (resp. higher) than the priority of τ_i is denoted $lp(\tau_i)$ (resp. $hp(\tau_i)$). In this paper, we denote R_i the response time of a task τ_i and B_i

the blocking factor incurred by the task τ_i. We also denote A_i the value of *allowance* on the execution duration of a task τ_i. The definition of allowance is given is Section 2.4.

2.2 Partitioned scheduling

The partitioned scheduling involves a static allocation of the tasks to the processors. The inter-processor migrations are not allowed and each processor can be scheduled independently. The problem of finding a feasible partition for a given set of tasks is a NP-hard problem because an instance of BIN-PACKING problem can be reduced in polynomial-time to an instance of task partitioning problem [18]).

Since the BIN-PACKING problem has been well studied, various approaches have been investigated to solve it. Several of these approaches can be used to find a solution to the partitioned scheduling problem. One of the famous approaches is the approximation algorithms. The well-known approximation algorithms FF, Best-Fit (BF) or NF are often used to implement partitioned scheduling algorithms. Other approaches such as dynamic programming has also been investigated by Baruah and Fisher to find an optimal solution to the partitioned scheduling problem [5]. Actually, many techniques for optimization problems can be adapted to solve the partitioned scheduling problem. For instance, Hou *et al.* applied the genetic algorithm to the multiprocessor scheduling [19], Tindell *et al.* [28] and Di Natale and Stankovic [14] applied the simulated annealing to find a feasible partition of a set of tasks. We describe in Section 3.2 how we have used the simulated annealing to find a feasible partition for which the robustness to the WCET overruns faults is maximized.

2.3 Shared resources

On a real platform, the tasks are not often independent. They performs accesses to the shared memory or to the input/output devices. To ensure the consistency of data during these accesses, a synchronization protocol is needed. Moreover, such a protocol must avoid deadlock situations. In real-time systems, the synchronization protocol must also avoid the unbounded priority inversions to ensure that a high-priority task is not delayed indefinitely by low-priority tasks.

In uniprocessor scheduling, the *Priority Ceiling Protocol* (PCP) has been proposed by Sha *et al.* [27]. This protocol bounds the priority inversions and avoids deadlocks. Rajkumar *et al.* presented an extended version of PCP for real-time multiprocessor system under static-priority partitioned scheduling : *Distributed-Priority Ceiling Protocol* (D-PCP) [26]. D-PCP has been designed for the distributed systems and it cannot take advantage of multiprocessor systems with shared memory, therefore *Multiprocessor-Priority Ceiling Protocol* (M-PCP), which relies on globally shared memory, has been presented by Rajkumar [25].

In a recent work [21], a partitioning algorithm based on BF for the assignment of the tasks has been proposed. It is based on M-PCP for the synchronization protocol. This algorithm is said *synchronization-aware* in sense where it considers the blocking factor induced by the synchronization protocol in the assignment policy. The tasks are assigned to the processors in such a way that the scheduling penalties associated with global task synchronization are minimized.

As an alternative to PCP, Baker proposed the *Stack Resource Policy* (SRP) [2]. This protocol has been extended to *Multiprocessor-Stack Resource Policy* (M-SRP) by Gai *et al.* [17], who also presented an comparison between M-PCP and M-SRP. Their study shown that M-SRP outperforms M-PCP.

Block *et al.* introduce the *Flexible Multiprocessor Locking Protocol* (FMLP) and show it outperforms M-SRP [8]. This protocol is flexible because it can be used for global scheduling as well as for partitioned scheduling. Brandenburg and Anderson discussed a detailed description of the implementation of FMLP for a static-priority partitioned scheduling and gave a bound on the blocking time B_i incurred by a task τ_i in appendix of [10].

2.4 Robustness

The constraints in a hard real-time system are defined such that no deadlines of any task are missed. Moreover, the WCET of a task is estimated or computed in order to ensure that the task never runs for a duration longer than its WCET. If a task commits a WCET overruns fault, the system may fail unless this WCET overrun does not cause any deadline misses. The duration which can be added to the WCET of a task such that all tasks of τ meet their deadlines has been denoted allowance by Bougueroua *et al.* [9]. The greater the value of allowance on the execution duration for each task is, the more the system robust to the WCET overruns faults is. Davis and Burns proposed a priority assignment which maximizes the allowance on the execution duration of all the tasks

of a system in uniprocessor [12]. The priority assignment is the algorithm which assigns a static priority to each task. In this work, we consider a multiprocessor system and we deal with both the priority assignment and the allocation of the tasks.

In our approach, we need to compute the allowance on the execution duration of the tasks. We have extracted from the literature two methods to compute this duration. The first one is based on the Response Time Analysis (RTA) and has been proposed by Bougueroua *et al.* [9]. For a given value A_i of allowance on the execution duration of a task τ_i, it consists in verifying that:

- the total CPU utilization does not exceed 1 (Equation (1)),
- the response time R_i of τ_i does not exceed its deadline D_i (Equation (2)),
- the response times R_j of all the low-priority tasks τ_j ($P_j < P_i$) do not exceed their deadlines D_j (Equation (3)).

$$u_{sum}(\pi_j) + \frac{A_i}{T_i} \leq 1 \tag{1}$$

$$R_i^{k+1} = C_i + A_i + \sum_{\tau_h \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_h} \right\rceil C_h \leq D_i \tag{2}$$

$$\forall \tau_l \in lp(\tau_i),$$
$$R_l^{k+1} = C_l + \sum_{\tau_h \in hp(\tau_l)} \left\lceil \frac{R_l^k}{T_h} \right\rceil C_h + \left\lceil \frac{R_l^k}{T_i} \right\rceil A_i \leq D_l \tag{3}$$

We know that the deadline of a task must not be missed and the total CPU utilization cannot exceed 1. Then a bound A_i^{max} on the maximum value of allowance on the execution duration for the task τ_i is given by $min(D_i - C_i, \lfloor (1 - u_{sum}(\pi_j))T_i \rfloor)$. For a task τ_i, the value of allowance on the execution duration is computed by searching a value A_i which verifies Equation (1), (2) and (3) (e.g. with a binary search).

The second method to compute the allowance on the execution duration is based on the *Sensitivity Analysis* (SA) proposed by Bini *et al.* [7]. This method is used to determine the feasibility region related to the variation of the WCET parameter. It applies on a static-priority scheduling. The value ΔC_i is the sensitivity on the WCET of the task τ_i and is given by Equation (4):

$$\Delta C_i = \min_{j \in lp(i)} \max_{t \in schedP_j} \frac{t - \left(C_j + \sum_{h \in hp(j)} \left\lceil \frac{t}{T_h} \right\rceil C_h \right)}{\lceil t / T_i \rceil} \tag{4}$$

where $schedP_j$ is the set of schedulable instants defined by $schedP_j = \mathcal{P}_{j-1}(D_j)$ and $\mathcal{P}_j(t)$ is defined by:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_j(t) = \mathcal{P}_{j-1}(\left\lfloor \frac{t}{T_j} \right\rfloor T_j) \cup \mathcal{P}_{j-1}(t) \end{cases} \tag{5}$$

The value ΔC_i is the maximum duration such that if a task set $\tau = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$ is schedulable, then the task set $\tau' = \{\tau_1, \ldots, \tau_i', \ldots, \tau_n\}$ is schedulable with $C_i' = C_i + \Delta C_i$. It corresponds to the definition of allowance given previously and $A_i = \Delta C_i$.

In order to implement our solution, we have chosen the first method based on the RTA since the Equations (2) and (3) can be simply changed in:

$$R_i^{k+1} = C_i + B_i + A_i + \sum_{\tau_h \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_h} \right\rceil C_h \leq D_i \tag{6}$$

$$\forall \tau_l \in lp(\tau_i),$$
$$R_l^{k+1} = C_l + B_l + \sum_{\tau_h \in hp(\tau_l)} \left\lceil \frac{R_l^k}{T_h} \right\rceil C_h + \left\lceil \frac{R_l^k}{T_i} \right\rceil A_i \leq D_l \tag{7}$$

3 Solution description

3.1 FMLP

We decide to implement FMLP in our solution because it outperforms all the previous synchronization protocols [8]. Furthermore, it has been adapted to the partitioned scheduling by the authors of [10].

In FMLP, two types of resources are considered: the short and the long resources. The resources are classified by the designer of the application. When a task τ_i tries to lock a long resource R_l, there are two eventualities:

- if R_l is already locked then τ_i suspends and it gets the highest priority when resuming,
- otherwise, τ_i locks R_l and runs non-preemptively.

Now, if the task τ_i tries to lock a short resource R_s, the two possibilities are:

- R_s is already locked then τ_i busily waits until R_s is unlocked,
- otherwise, τ_i locks R_s and runs non-preemptively.

In Figure 2, we represent an example comprised of four tasks scheduled on two processors and synchronized by the FMLP protocol. The jobs of τ_1 and τ_2 are scheduled on the first processor and the jobs of τ_3 and τ_4 are scheduled on the second one. A short resource (vertically hatched) is shared by τ_1 and τ_4 and a long resource (horizontally hatched) is shared by τ_2 and τ_3. At time 0, τ_2 starts and at time 3, it locks the long resource. At time 2, τ_3 starts and suspends at time 4 since the long resource is already locked by τ_2. At time 5, τ_4 starts because τ_3 is suspended, it locks the short resource at time 7 and continues its execution non-preemptively. At time 6, τ_1 is activated but cannot be scheduled since τ_2 runs non-preemptively. At time 8, τ_2 unlocks the long resource and τ_1 can start but τ_3 cannot lock the long resource because τ_4 is always running. τ_1 is blocked at time 9 because τ_4 has locked the short resource and it busily waits until time 10. At time 10, τ_4 unlocks the short resource and both τ_1 which was blocked by the lock and τ_3 which was deferred by the execution of τ_4 can be executed.

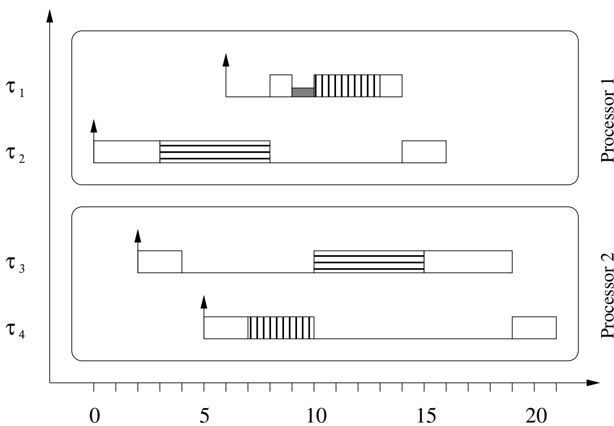


Figure 2. Four jobs on two processors and two resources synchronized by FMLP.

In order to analyze the feasibility of a set of tasks, we need to bound the blocking time incurred by the tasks. Each task τ_i is subjected to various types of blocking:

- *boost blocking* denoted BB_i and which is incurred when a lower-priority task on the same processor is sharing a long resource,
- *arrival blocking* denoted AB_i and which is incurred when a lower-priority task on the same processor is sharing a short resource,
- *short blocking* denoted SB_i and which is incurred when a task on another processor is locking a short resource,
- *long blocking* denoted LB_i and which is incurred when a task on another processor is locking a long resource,
- *deferral blocking* denoted DB_i and which is incurred when a higher-priority task on the same processor defers its execution.

The blocking factor B_i of a task τ_i is given by the relation $B_i = BB_i + AB_i + SB_i + LB_i + DB_i$. A bound for each of these factor is given in appendix of [10]. The blocking factor B_i has been implemented in our algorithm in order to provide a feasibility analysis which takes into account the interferences due to priority inversions.

3.2 Simulated Annealing

Simulated annealing is a generic algorithm which has been firstly proposed by Kirkpatrick *et al.* [20] for the optimization problems. The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and to reduce their defects.

Tindell *et al.* [28] has used simulated annealing to find a feasible allocation of the tasks in a distributed hard real-time system. Di Natale and Stankovic [14] have also used this technique in distributed systems where the tasks are periodic and have arbitrary deadlines, precedence and exclusion constraints.

In this work, we apply the simulated annealing technique to build an algorithm which finds a feasible partition of a set of tasks where the robustness to the WCET overruns faults is maximized.

We describe this algorithm, *Robust Scheduler based on Simulated Annealing* (RSSA) in Algorithm 1. The initialization is made as follows. At line 1, the function `random_partition()` build a partition P by allocating each of the n tasks on one of the m processors randomly. This partition may be

unfeasible. At line 2, an initial temperature is computed such as 99% of the partitions are kept even if they do not improve the solution. By cooling the system (decreasing the temperature), the unsatisfying solutions will be eliminated. At line 3, we initialize the `max_try` value to an integer which depends both of the number of tasks and of the number of processors. The loop at line 4-21 performs `max_try` iterations of the loop at line 6-19. After each iteration, the system is cooled by dividing the temperature by 2.

The main part of the algorithm is the loop at line 6-19. At line 7, the function `compute_energy()` computes the energy of the partition P. This function is more detailed later in Algorithm 2. At line 8, a partition P_n which is the neighbor of the partition P is computed. This partition P_n is obtained either by randomly swapping two tasks of P or by randomly moving a task of P from a processor to another. The energy of this new partition is computed at line 9. If the value E_n of the energy of P_n is less than the value E_p of the energy of P then P is replaced by P_n. Otherwise, a random number is drawn between 0 and 1. The more the temperature $temp$ high is, the more the probability that the value e^x ($x = \frac{E_p - E_n}{temp}$) is greater than the random number is. If $e^x > random(0,1)$ then P is also replaced by P_n else P_n is discarded. This behavior avoids that the energy converge to a local minimum.

We now describe the function `compute_energy()`. The aim of this function is to compute a value of energy for a partition such that the more the minimum value of allowance for the system is great, the less the value of energy is. The value of energy is computed as follows. For each processor π_j, if the processor is empty then the value of energy is increased by 1. This behavior increase the probability that the tasks are well distributed among the processor and no processors stay empty. If the set of tasks allocated on this processor is unschedulable then the value of energy is also increased by 1 to eliminate the unfeasible partitions. For each processor π_j where the set of tasks is schedulable, the sum of allowance on the execution duration values of each task is stored in the array `allowance` at index j. At the end of the loop at line 3-10, the value of energy is increased by the sum of all the values stored in the array `allowance`. Consequently the more the value of allowance of each task great is, the less the value of energy is.

Algorithm 1: Robust Simulated Annealing

1 $P = random_partition(n, m)$;
2 $temp = \frac{-m}{ln(0.99)}$;
3 $max_try = n \cdot m$;
4 **while** $temp > 10^{-5}$ **do**
5     $k = 0$;
6   **while** $k \neq max_try$ **do**
7         $E_p = compute_energy(P)$;
8         $P_n = neighbor(P)$;
9         $E_n = compute_energy(P_n)$;
10   **if** $E_n < E_p$ **then**
11             $P = P_n$;
12   **else**
13             $x = \frac{E_p - E_n}{temp}$;
14   **if** $e^x \geq random(0,1)$ **then**
15                 $P = P_n$;
16   **end**
17   **end**
18         $k = k + 1$;
19   **end**
20     $temp = \frac{temp}{2}$;
21 **end**

Algorithm 2: compute_energy(P)

1 $energy = 0$;
2 $allowance[m]$;
3 **foreach** $\pi_j \in P$ **do**
4   **if** π_j *is empty or* π_j *is unschedulable* **then**
5         $energy = energy + 1$;
6         $allowance[j] = 0$;
7   **else**
8         $allowance[j] = \sum_{\tau_i \in \tau(\pi_j)} A_i$
9   **end**
10 **end**
11 $energy = energy + \frac{1}{\sum_{k=1}^{m} allowance[k]}$

4 Simulation

4.1 Methodology

We implemented a scheduling simulator with several partitioned scheduling algorithms. We implemented in this simulator a partitioned scheduling algorithm based on simulated annealing and a builder of partitioned scheduling algorithms based on approximation algorithms. The input parameters of the

builder are the number of processors, the approximation algorithm (FF, NF or custom algorithms), a uniprocessor scheduling algorithm and the feasibility condition. We also implemented a generator which randomly generates tasks sets with shared resources.

We consider here a multiprocessor platform comprised of 8 identical processors. The sets of tasks are generated using a method similar to the one given in [6]. We extract a set τ^r of $m+1$ randomly generated tasks. τ^r is accepted if its utilization is less than the total CPU utilization ($u_{sum}(\tau^r) < m$). This feasibility condition is a necessary condition. While $u_{sum}(\tau^r) < m$, we generate another task which is added to τ^r. This new set is accepted if the previous given condition is respected. When a set has an utilization which does not respect the feasibility condition, it is rejected and we extract a new set of tasks with $m+1$ tasks.

We randomly generated 10,000 sets of tasks with constrained deadlines by applying the given method of generation. The utilization of each task τ_i is normally distributed with a mean of 0 and a standard variation of 0.25. The tasks with a negative utilization or with an utilization greater than 1 are discarded. The periods are uniformly distributed between 1 and 2000 and the WCETs are extracted from the relation $C_i = u_i \cdot T_i$. The deadlines are uniformly distributed between C_i and T_i.

For each set of tasks, we randomly generated $\frac{n}{m}$ resources which are either short or long. The probability that a resource is short or long is 50%. The duration of a short resource is uniformly distributed between 1 and 10 and the duration of a long resource is uniformly distributed between 11 and 50.

A simulation consists in applying three partitioning algorithms on each set of tasks. A platform constituted in four processors is considered in this simulation. The first partitioned scheduling algorithm is our RSSA algorithm based on simulated annealing.

The two last algorithms are based on approximation algorithms for the BIN-PACKING problem. These algorithms are built in the following manner. The tasks are sorted in a decreasing utilization order. The uniprocessor scheduling algorithm on each processor is DM. A task τ_i can be allocated to a processor π_j if its Worst-Case Response Time (WCRT) and the WCRT of each task already allocated on π_j is less than or equal to its deadline. This WCRT is computed by RTA extended by taking into account the blocking factor of the tasks. Actually, the computation of the allowance on the execution duration of the tasks (based on the RTA) is used as a feasibility condition. The first algorithm is built from the FF algorithm which is know to offer good results in terms of schedulability. The FF algorithm allocates a task on the first processor which can accept it. The second one is built from the *Worst-Fit* algorithm. This algorithm allocates a task on the processor which has the smallest value of utilization.

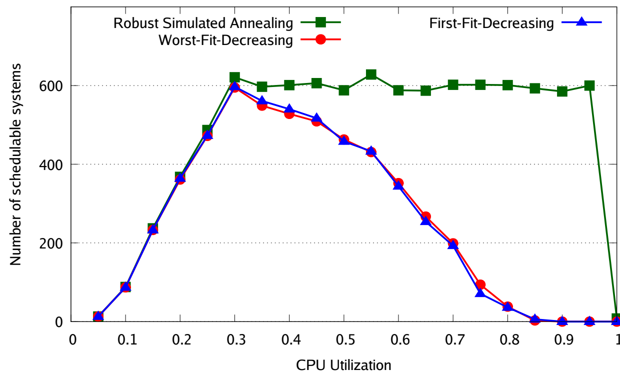
4.2 Results

Our simulator produces a set of log trace for each set of tasks. From these traces, we extracted three main results. The first one is the performance of the algorithms in term of schedulability. We show in Figure 3(a) the number of feasible systems for each value of CPU utilization U_i ($U_i \in [0.05, 0.1, 0.15, \ldots, 1]$) regarding to the chosen partitioned scheduling algorithm. Our algorithm based on simulated annealing success in finding a feasible partition for 8384 tasks sets (i.e. 84% of the generated tasks sets). The WF, respectively FF, found 5194, respectively 5180, feasible tasks sets (i.e. 52%). We notice that the performance in terms of schedulability of WF and FF are similar. But it has been seen in [15] that FF outperforms WF in terms of schedulability when the tasks are independent. In Figure 3(a), we show that the performance of WF and FF are slightly equivalent. This result is due to the fact that the blocking factor of the tasks becomes a constraint at least as important as the utilization of the tasks.

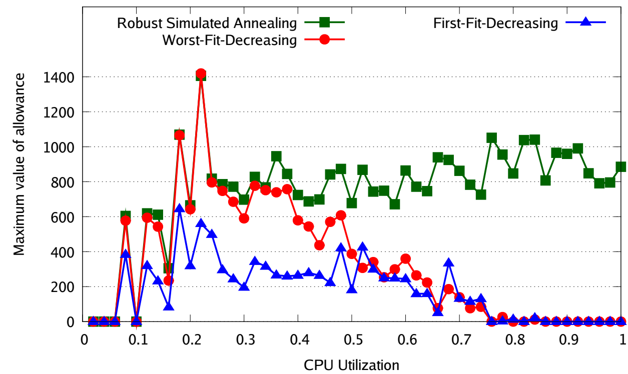
The second one is the value of maximum allowance on the execution duration for each value of CPU utilization. This value of allowance is obtained by taking the average value for all the sets of tasks which have the same value of CPU utilization. We show in Figure 3(b) that WF outperforms FF in terms of minimum allowance. This results has already been seen in [15]. We also notice that from CPU utilization under 40%, the minimum allowance value is equivalent to WF. But over 40%, our RSSA algorithm outperforms WF.

The last one is the value of minimum allowance on the execution duration and these values are represented in Figure 3(c). We notice that RSSA also outperforms WF for a CPU utilization over 50%. Indeed, we show that the result shown in Figure 3(b) is not biased by allocating a task alone on a processor to maximize the value of maximum allowance.

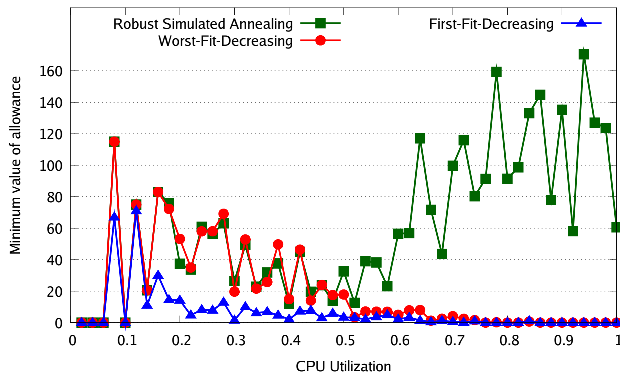
We show from these results that the simulated annealing approach is an interesting solution when sev-



(a) Schedulability



(b) Maximum allowance



(c) Minimum allowance

Figure 3. Simulations with 4 CPU.

eral criteria must be optimized. In our case, we have intended to maximize both the schedulability and the robustness of the system.

5 Conclusion

We have proposed a partitioned scheduling algorithm (RSSA) based on the technique of simulated annealing. We have considered a model of sporadic tasks with shared resources. The consistency of data is ensured by the FMLP protocol. Our algorithm allocates the tasks on the processors in order to maximize the robustness to the WCET overruns faults of each task. We have implemented our algorithm in a scheduling simulator and we have shown that this solution outperforms the scheduling algorithms based on approximation algorithms in terms of both schedulability and robustness. In order to improve this work, we intend to compare RSSA with other approaches and in particular with an optimal one.

References

[1] B. Andersson and J. Jonsson. Preemptive multiprocessor scheduling anomalies. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 12–19, Fort Lauderdale, Florida, USA, April 2002. IEEE Computer Society.

[2] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[3] S. K. Baruah and J. Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *Proceedings of the 15th Euromicro Conference on Real-time Systems (ECRTS)*, pages 195–202, Porto, Portugal, July 2003. IEEE Computer Society.

[4] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[5] S. K. Baruah and N. W. Fisher. A dynamic programming approach to task partitioning upon memory-constrained multiprocessors. In *Proceedings of the 10th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, page 9pp, Gothenburg, Sweden, August 2004. Springer-Verlag.

[6] M. Bertogna. Evaluation of existing schedulability tests for global EDF. In *Proceedings of the 38th International Conference on Parallel Processing*

Workshops (ICPPW), pages 11–18, Vienna, Austria, September 2009. IEEE Computer Society. First International Workshop on Real-time Systems on Multicore Platforms: Theory and Practice (XRTS).

[7] E. Bini, M. Di Natale, and G. C. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *Proceedings of the 18th Euromicro Conference on Real-time Systems (ECRTS)*, pages 13–22, Dresden, Germany, April 2006. IEEE Computer Society.

[8] A. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson. A flexible real-time locking protocol for multiprocessors. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47–56, Daegu, South Korea, August 2007. IEEE Computer Society.

[9] L. Bougueroua, L. George, and S. Midonnet. Dealing with execution-overruns to improve the temporal robustness of real-time systems scheduled FP and EDF. In *Proceedings of the 2nd International Conference on Systems (ICONS)*, page 8pp, Sainte-Luce, Martinique, April 2007. IEEE Computer Society.

[10] B. B. Brandenburg and J. H. Anderson. An implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP real-time synchronization protocols in $LITMUS^{RT}$. In *Proceedings of the 14th IEEE International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, pages 185–194, Kaohsiung, Taiwan, August 2008. IEEE Computer Society.

[11] H. Cho, B. Ravindran, and E. D. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)*, pages 101–110, Rio de Janeiro, Brazil, December 2006. IEEE Computer Society.

[12] R. I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS)*, pages 3–14, Tucson, Arizona, USA, December 2007. IEEE Computer Society.

[13] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, January-February 1978.

[14] M. Di Natale and J. A. Stankovic. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS)*, pages 190–199, Pisa, Italy, December 1995. IEEE Computer Society.

[15] F. Fauberteau, S. Midonnet, and L. George. A robust partitioned scheduling for real-time multiprocessor systems. In *Proceedings of IFIP Conference on Distributed and Parallel Embedded Systems (DIPES)*, page (to appear), Brisbane, Australia, September 2010. Springer Science and Business Media.

[16] N. W. Fisher, S. K. Baruah, and T. P. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proceedings of the 18th Euromicro Conference on Real-time Systems (ECRTS)*, pages 118–127, Dresden, Germany, July 2006. IEEE Computer Society.

[17] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 189–198, Toronto, Canada, May 2003. IEEE Computer Society.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[19] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, February 1994.

[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[21] K. Lakshmanan, D. de Niz, and R. R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, pages 469–478, Washington, D.C., USA, December 2009. IEEE Computer Society.

[22] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

[23] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):47–61, 1973.

[24] A. K. Mok. *Fundamental Design Problems of Distributed Systems for Hard-Real-Time Environments*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, May 1983.

[25] R. (Raj) Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings of the 10th International Conference on Distributed Computing Systems (ICDCS)*, pages 116–123, Paris, France, May-June 1990. IEEE Computer Society.

[26] R. (Raj) Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS)*, pages 259–269, Huntsville, AL , USA, December 1988. IEEE Computer Society.

[27] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[28] K. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.