Robust Partitioned Scheduling for Real-Time Multiprocessor Systems

Frédéric Fauberteau, Serge Midonnet, Laurent George

Robust Partitioned Scheduling for Real-Time Multiprocessor Systems

Frédéric Fauberteau[1], Serge Midonnet[1], and Laurent George[2]

[1] Université Paris-Est
LIGM, UMR CNRS 8049
5, bd Descartes, Champs-sur-Marne, 77454 Marne-la-Vallée CEDEX 2, France
{fauberte,midonnet}@univ-mlv.fr
[2] ECE / LACSC
37,quai de Grenelle, 75015 Paris, France
lgeorge@ieee.org

Abstract. In this paper, we consider the problem of fixed-priority partitioned scheduling of sporadic real-time tasks for homogeneous processors. We propose a partitioning heuristic that takes into account possible Worst Case Execution Time (WCET) overruns. Our goal is to maximize the duration a task can be allowed to exceed its WCET without compromising the timeliness constraints of all the tasks. This duration is denoted in the paper the *allowance* of the task and is computed with a sensitivity analysis. The partitioning heuristic we propose, assigns the tasks to the processors in order (i) to maximize the *allowance* of the tasks and (ii) to tolerate bounded execution duration overruns. Property (ii) is important as real-time applications are often prone to be subject to OS approximations or software faults that might result in execution duration overruns. We show with performance evaluations that *Allowance-Fit-Decreasing* partitioning improves the temporal robustness of real-time systems w.r.t. classical {*First-Fit*/*Best-Fit*/*Next-Fit*}-*Decreasing* partitioning.

Key words: Real-time Scheduling, Partitioned Scheduling, Robustness

1 Introduction

Fixed-priority scheduling of recurring real-time tasks has been largely studied for uniprocessors. In such a scheduling, a Priority Assignment (PA) assigns a fixed priority to each job of the task. For instance, Rate-Monotonic (RM) is an optimal PA for periodic tasks with *implicit-deadlines* (deadlines equal to periods) [1]. Optimality implies that if a feasible PA over a taskset exists, then the optimal PA is also feasible. A feasible taskset is a taskset such that a scheduling algorithm exists which can schedule this taskset. We focus on the more general model of tasks with *constrained-deadlines* (deadlines less than or equal to periods) for which Deadline-Monotonic (DM) is an optimal PA [2]. Recently, the optimal Robust Priority Assignment (RPA) [3] has been proposed to find the PA which maximizes the interference that a tasks system can support. These interferences

can be handled by the tasks by allowing WCETs overruns while the timeliness constraints of all the tasks are respected. These tolerated WCETs overruns are denoted *allowance* of the tasks. In the same way, our motivation is to propose a robust multiprocessor scheduling which maximizes the *allowance*.

The two most studied approaches to schedule real-time tasks on a multiprocessor are *partitioned* and *global* scheduling. The first one does not allow tasks to migrate whereas the second one allows unrestricted migrations. Recent architecture have reduced the cost of migration. Nevertheless, taking into account the cost of migration in the feasibility conditions of global scheduling is still an open issue. A recent performance evaluation of partitioned and global schedulings show that partitioned scheduling outperform global scheduling, in the current state-of-the-art of feasibility conditions [4]. We therefore focus on the partitioned approach. Several algorithms for fixed-priority partitioned scheduling have been proposed [5–9]. The aim of the authors is to propose algorithms which improve the worst-case *utilization bound*. The worst-case utilization bound for a scheduling algorithm A is defined as the minimum utilization for which any *implicit-deadline* taskset is schedulable according to algorithm A. The utilization of an *implicit-deadline* taskset is the sum of the processor utilization (formally defined in Sect.2) of each task composing this taskset.

In this paper, our motivation is slightly different since we want to design a partitioned scheduling which improves the temporal robustness of a system i.e. to improve its capability to support variations on the system parameters at run time (WCET overruns for e.g.). Such events should not lead to a deadline violation in a hard real-time application. We focus on the WCET parameter and we propose an algorithm which allocates the tasks on the processor having the greatest capability to support WCETs overrun by maximizing the minimum *allowance* of all the tasks.

The rest of this paper is organized as follows. In Section 2, we introduce the terminology used in the rest of this paper. In Section 3, we give a definition of robustness in context of this paper. In Section 4, we discuss two manners to compute the allowance of the execution duration which is the criterion of our partitioning algorithm for the assignment of real-time tasks on the processors. In Section 5, we present our heuristic and describe how it works. In Section 6, we compare on some simulations the performance of the partitioning schedulings and we explain the benefits of our approach. We summarize the contributions of this paper in Section 7 and we give direction for our future work.

2 Terminology

In this paper, 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 upper bound on the separation between release times of the jobs is known. Each task τ_i is characterized by a minimum interarrival time T_i (also denoted period), a worst-case execution time C_i and a relative deadline D_i. This application runs on a platform $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ of m identical processors (homogeneous

case). We consider a fixed-priority scheduling on each processor. A fixed-priority scheduler assigns a priority to each task and all jobs of a task is released with the fixed priority of this task. We assume that tasks are indexed by decreasing priority: $\forall i = 1, \ldots, n-1$, task τ_i has a higher priority than task τ_{i+1}. A partitioning algorithm produces a partition $Part(\tau) = \{\tau^1, \tau^2, \ldots, \tau^m\}$ of m disjointed subsets where each subset τ^j of real-time tasks is executed on processor π_j. The subset τ^j composed by n_j tasks is also denoted by $\tau^j = \{\tau_1^j, \tau_2^j, \ldots, \tau_{n_j}^j\}$. In the rest of this paper, we refer to τ_i when the considered task is taken independently and to τ_i^j when it is considered assigned on processor π^j. We define u_i as the utilization of task $\tau_i : u_i \equiv \frac{C_i}{T_i}$ and U^j as the utilization of the taskset τ^j : $U^j \equiv \sum_{\tau_k^j \in \tau^j} u_k$. On the processor π_j, we denote $lp^j(i)$ (respectively $hp^j(i)$) the subset of real-time tasks assigned to π^j which have a priority lower than (respectively higher than or equal to) τ_i. The response time of the task τ_i is denoted R_i. We denote R_i^k the k^{th} iteration in the response time computation of the task τ_i.

3 Robustness

We consider the robustness in the real-time systems as the capacity of the system to handle WCET overruns faults when the WCET are estimated. If the WCET of all the tasks of the system has been well defined, a feasibility analysis shows wheter the system is feasible. But in pratice, it may possible that a task makes a fault or that the time constraints has been miscalculated. Some real-time specifications - such as Real-Time Specification for Java [10] - provide mechanism to handle cost overruns and deadline misses in the case of estimated WCET.

In this work, we consider the robustness as the capacity of a system to meet all the deadlines. We can guarantee that the system stay feasible if and only if we know the execution duration during which a task can exceed its WCET without any deadline is missed. This duration is denoted *allowance* and the more each task *allowance* has, the more robust the system regarding to our definition is.

4 Allowance concept

The allowance of a task is used as a criterion for allocating a task on a processor by our heuristic. We define the *allowance* A_i^j of a task τ_i^j on the processor π^j as follows :

Definition 1. *Let τ^j be a given set of tasks assigned on processor π^j. The allowance A_i^j of a task τ_i^j of τ^j is the maximum duration which can be added to the WCET C_i of τ_i^j such as all tasks of τ^j meet their deadlines.*

We identified in the literature two approaches to compute the *allowance*: one based on a (Worst Case Response Time) WCRT computation and one based on a sensitivity analysis on the WCETs. The response time of a task is the duration between the time this task has been released and the time it has been

finished. The WCRT of a task is the response time of this task in the worst activation scheme. We use the taskset given in Tab.1 to describe the two *allowance* computation methods in the following subsections.

Table 1. System of 4 sporadic real-time tasks.

	C_i	D_i	T_i
τ_1	10	60	70
τ_2	15	85	100
τ_3	30	190	210
τ_4	45	260	320

4.1 Allowance Computed from WCRT

One of the available approach to compute the allowance of the execution duration has been proposed by Bougueroua *et al.* [11]. For a given value of allowance A_i^j, this method consists in checking that the system remains schedulable when the execution duration of a task τ_i is equal to $C_i' = C_i + A_i^j$. In other words, this method consists in checking that the WCRT of all the tasks remains less than or equal to their deadline when their WCET is extended to $C_i + A_i^j$. The 3 following equations perform this check for the task τ_i^j on a processor π_j if τ_i^j was assigned on τ^j.

$$U^j + \frac{A_i^j}{T_i} \leq 1 \tag{1}$$

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

$$\forall \tau_l \in lp^j(i),$$

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

The value of allowance of a real-time task τ_i is found by a binary search. Equation (1) tests if the utilization U^j of the system when the WCET of τ_i^j is extended to $C_i + A_i^j$ does not exceed processor utilization. An upper bound $A_{i,up}^j$ on the *allowance* of the task τ_i can be found from (1):

$$A_{i,up}^j = \left\lfloor (1 - U^j) \cdot T_i \right\rfloor \tag{4}$$

Equation (4) allows to bound the binary search in $[0, A_{i,up}^j]$. For the task τ_1^j of our example, $A_{1,up}^j = \lfloor (1 - 0.58) \cdot 70 \rfloor = 29$. We can carry out a binary search with $0 \leq A_1 \leq 29$. Equation (2) tests if the response time R_1 of τ_1,

when its WCET has been extended to $C_1 + A_{1,j}$, does not exceed its deadline D_1. Equation (3) tests if the response times R_l of all tasks τ_l^j of lower priority than τ_1^j don't exceed their deadlines D_l when the WCET of τ_1^j is extended by $A_{1,j}$. For the value $\lfloor 29/2 \rfloor = 14$, we must check that this value satisfies (2) and (3). For $A_{1,j} = 14$, the response time of τ_1^j is $R_1 = 24 \leq D_1$. We also obtain $R_2 = 39 \leq D_2$, $R_3 = 69 \leq D_3$ and $R_4 = 177 \leq D_4$. Then $A_{1,j} = 14$ is a valid value of allowance for τ_1^j on the processor π^j. We continue the binary search until $A_{1,j} = 21$, then $A_1 = 21$ is the maximum value of allowance for $\tau_{1,j}$.

The complexity of this approach is pseudo-polynomial due to the Response Time Analysis (RTA) in (2) and in (3). This complexity is in $O(n^2)$ where n is the number of tasks. Indeed, for a task τ_i, a RTA is performed in $O(n)$ and for each task of lower priority than τ_i, a RTA is performed in $O(n)$. In the worst case, there is $n-1$ tasks of lower priority than τ_1, thus the complexity is $O(n^2)$.

4.2 Allowance Computed from Sensitivity Analysis

Another approach to compute the allowance of the execution duration is the *sensitivity analysis*. This approach has been introduced by Bini *et al.* [12]. This approach is attractive compared to the previous one because no iterative computation (such as WCRT computation) is needed. The authors propose to consider the system only at time corresponding to the activation time of the highest priority tasks in $[0, D_i]$ union time $\{D_i\}$. The maximum allowance A_i^j of a task τ_i^j on the processor π^j is computed by the following equations:

$$Sens(k) = \max_{t \in schedP_k} \frac{t - \left(C_k + \sum_{h \in hp(k)} \left\lceil \frac{t}{T_h} \right\rceil C_h \right)}{\lceil t/T_i \rceil} \tag{5}$$

$$A_i^j = \left\lfloor \min_{k \in lp(i)} Sens(k) \right\rfloor \tag{6}$$

where $schedP_k$ is the set of scheduling points defined by $schedP_k = \mathcal{P}_{i-1}(D_k)$ and $\mathcal{P}_k(t)$ is defined by :

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_k(t) = \mathcal{P}_{k-1}(\left\lfloor \frac{t}{T_k} \right\rfloor T_k) \cup \mathcal{P}_{k-1}(t) \end{cases} \tag{7}$$

For the task τ_1 of our example, $schedP_1 = D_1 = \{60\}$. In the same way, $schedP_2 = \{70, 85\}$, $schedP_3 = \{70, 100, 140, 190\}$ and $schedP_4 = \{140, 200, 210, 260\}$. The values $Sens(1) = 50$, $Sens(2) = 45$, $Sens(3) = 33.33$ and $Sens(4) = 21.66$ are computed by using Equation 5. The *allowance* $M_1 = \min_{k \in lp^j(i)} \lfloor Sens(k) \rfloor = 21$ is obtained from (5) and (6).

The complexity of this approach is exponential because $|schedP_n|$, denoted as the size of the scheduling points set computed by (7) is 2^{n-1} in the worst-case. But in practice, the size of $schedP_n << 2^{n-1}$ for a great value of n. We notice that the size of $schedP_n$ is highly sensitive to the range of task periods as seen in appendix of [13].

5 Partitioning Algorithm

5.1 Task Partitioning Problem

The task partitioning problem consists of finding a partition of a taskset τ in m subsets τ^j, $1 \leq j \leq m$ such that each subset is feasible on processor π_j. Since it has been proved that BIN-PACKING problem (NP-hard in the strong sense) can be reduced in polynomial-time to a task partitioning problem [14], no optimal algorithm exists to decide in polynomial-time if a given taskset is feasible. Fortunately, approximation algorithms and heuristics exist to find solutions for the task partitioning problem in polynomial-time. Heuristics for the tasks partitioning problem exist and are versions of the heuristics proposed for the bin-packing problem. The more cited in the literature are *First-Fit* [8] (FF), *Best-Fit* [6] (BF) and *Next-Fit* (NF) [7]. These heuristics have been initially designed to minimize the number of bins (respectively the number of processors) for the BIN-PACKING problem (respectively the tasks partitioning problem). Another heuristic *Worst-Fit* [15], is rarely used because it provides poor performance to solve the BIN-PACKING problem. On the problem of task partitioning, this heuristic allocates tasks to processors where utilization is the lowest. This approach is relevant because we want the best allocation of tasks to maximize the *allowance* of tasks. We propose in the next subsection a heuristic which allocates tasks to processors that has the greatest *allowance* rather than processors that has the lowest utilization.

5.2 Allowance-Fit-Decreasing

We propose a heuristic, denoted *Allowance-Fit-Decreasing*, to solve the task partitioning problem. We want to tolerate bounded WCET overruns, a property not considered in classical heuristics. WCETs overruns can be due to OS approximations, faults of the task or WCET under-estimation. By definition, the allowance of a processor is the minimum *allowance* for all task allocated to the processor. Our goal is to propose a partitioning scheme that assigns a task to the processor whose allowance is maximum.

We describe the *Allowance-Fit-Decreasing* heuristic with the pseudo-code given in Alg.1. The tasks are first sorted according to their utilization by function `sort_task_by_decreasing_utilization()` (line 1). For each task τ_i of the taskset τ (iteration loop at lines 2-17), the `proc` parameter, denoting the processor on which τ_i is allocated (at line 3), is initialized with a `null` value. The minimum value of *allowance* for the entire system (variable A_{min} at line 4) is first initialized to minus infinity. We then consider all processors in Π and find the processor that maximizes the processor *allowance*. For each processor π_j (iteration loop at lines 5-11), our heuristic finds the minimum value of processor *allowance* A^j_{min} computed by function `compute_allowance(`π^j`,` τ^j_i`)` when τ_i is allocated to π_j with the method described in Sect.4.2. If after the iteration loop, A_{min} is greater than or equal to 0 then, τ_i can be assigned to a processor, the one that maximizes the processor *allowance*, by construction. We then proceed

Algorithm 1: Allowance-Fit-Decreasing

```
1  sort_task_by_decreasing_utilization()
2  foreach τ_i in τ do
3  |   proc = None;
4  |   A_min = -Inf;
5  |   foreach π_j in Π do
6  |   |   A^j_min = compute_allowance(π_j, τ^j_i)
7  |   |   if A^j_min > A_min then
8  |   |   |   proc = π_j;
9  |   |   |   A_min = A^j_min;
10 |   |   end
11 |   end
12 |   if A_min ≥ 0 then
13 |   |   assign τ_i to proc
14 |   else
15 |   |   return unschedulable
16 |   end
17 end
18 return schedulable
```

with the other tasks until all the tasks until either all tasks have been assigned to a processor (we then return that the task set is schedulable in line 18) or one task is declared not schedulable (line 15).

5.3 Partitioned Scheduling Algorithm

A partitioned scheduling algorithm is the combination of a task partitioning algorithm with a schedulability condition. We build two partitioned scheduling algorithms, one from *Worst-Fit* and one from *Allowance-Fit-Decreasing*. For the first one, the schedulability condition is a necessary and sufficient condition implicitly given by *Allowance-Fit-Decreasing*. Indeed, our heuristic computes with the function `compute_allowance(`π_j`,`τ_i`)` the value of A^j_{min}, the minimum *allowance* for all the tasks assigned to processor π_j, including τ_i. If this function returns a negative value, then τ_i cannot be assigned on the processor π^j. A^j_{min} is computed from the sensitivity analysis given in Sect.4.2. For the second one, we combine *Worst-Fit* with the necessary and sufficient schedulability condition RTA [16].

During the allocation, the tasks are taken in order of their decreasing utilization. In other words, the tasks with the greater utilization are allocated first. We consider a fixed-priority assignment and we use DM priority assignment since this PA is an optimal one when the considered tasks have *constrained-deadlines* ($\forall i$, $D_i \leq T_i$) [2].

6 Simulations

6.1 Methodology

Our simulations is based on randomly generated tasksets. Because we focus on tasks with *constrained-deadlines*, we consider tasksets such that for any task τ_i, $\alpha = \frac{D_i}{T_i}$). We randomly generate 10 sets for $\alpha \in [0.1, 0.2, \ldots, 1.0]$. Each set is built from 100,000 randomly generated tasksets. Each taskset is composed by 24 tasks. A taskset is generated using the `UUniFast` algorithm [17] which produces a uniformly distributed set of task utilizations and which avoids bias in the generated tasksets. For a given task τ_i, the period T_i is generated with a uniform distribution between 100 and 100,000 ms. The deadline D_i is given by $D_i = \alpha \cdot T_i$ and the WCET C_i is given by $C_i = u_i \cdot T_i$. We consider an homogeneous processor composed by 8 identical processors.

6.2 Simulation Results

We show in Fig.1 the average number of iterations during the computation of *allowance*.

$$R_i^{k+1} = C_i + \sum_{\tau_h \in hp^j(i)} \left\lceil \frac{R_i^k}{T_h} \right\rceil C_h \tag{8}$$

We implement a function `iteration()` which computes the value given by (8). The computation of the *allowance* based on the WCRT computation calls this function in (2) and in (3). The computation of the *allowance* based on the sensitivity analysis calls this function in (5). For the two implementations of the *allowance* computation, we count the number of calls to the function `iteration()` for each randomly generated taskset and we keep the average of the *allowance* over all the taskset. We notice that despite the fact of the complexity of the sensitivity analysis seems greater than the complexity of the *allowance* computation based on the WCRT, the number of iterations for sensitivity analysis is well below the number of iterations for computation based on the WCRT. We therefore choose to compute *allowance* by sensitivity analysis [12].

We show in Fig.2, 3 and 5 the comparison between *First-Fit-Decreasing* (FFD), *Worst-Fit-Decreasing* (WFD) and *Allowance-Fit-Decreasing* (AFD). *Decreasing* means that these heuristics assign the tasks in order of their decreasing utilization. We voluntarily omit to show results concerning BF and NF because their behavior are very similar to FF. AFD and WFD use a necessary and sufficient condition of schedulability. Therefore we used the necessary and sufficient condition RTA [16] for the heuristics FFD.

We show in Fig.2 the number of partition found by the heuristics FFD, WFD and AFD. We notice that FFD provides a slightly better schedulability for $\alpha \geq 0.4$. This result is explained by the fact that FFD is one of the best heuristic for the task partitioning problem in terms schedulability. But the gap between FFD and the other two heuristics is not very large and may be acceptable if we want more robustness to WCETs overruns.

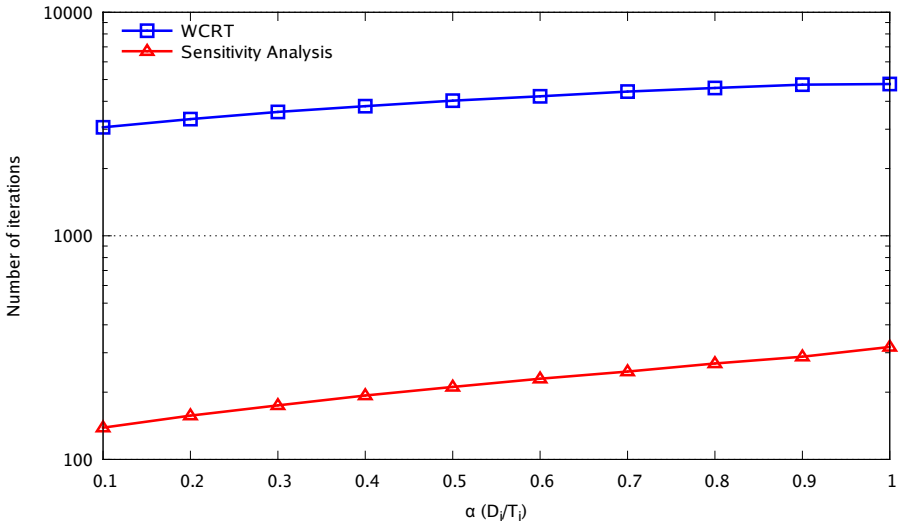

Fig. 1. Comparison of computation time for the *allowance* computation approaches.

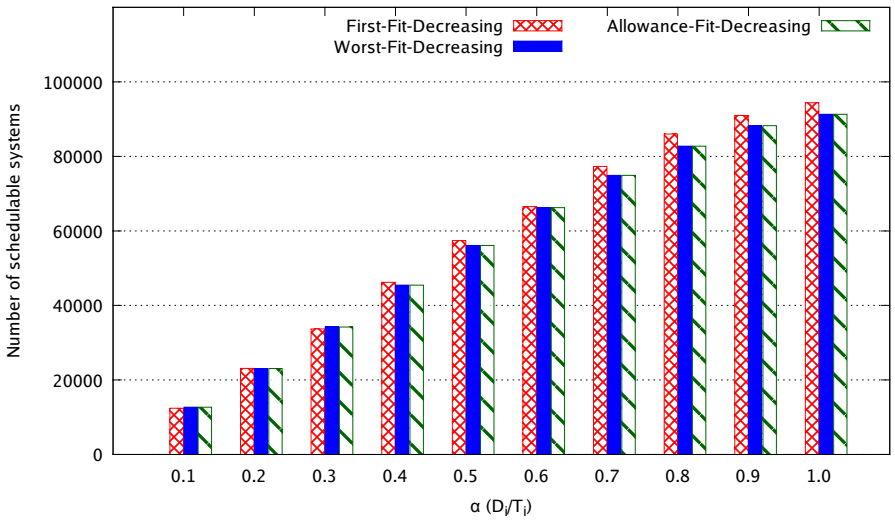

Fig. 2. Comparison of schedulability for the tasks partitioning heuristics.

We compare in Fig.3 the minimum *allowance* obtained by the three heuristics on 4 processors and in Fig.4 on 8 processors. Minimum *allowance* A_{min} guarantees that any task of the system can bear an interference during A_{min} without any deadline is missed. We show that AFD and WFD outperforms largely FFD.

Indeed, AFD and WFD distributes the tasks among the processors instead of fills up all the first processors. We note that AFD is slightly better than WFD.

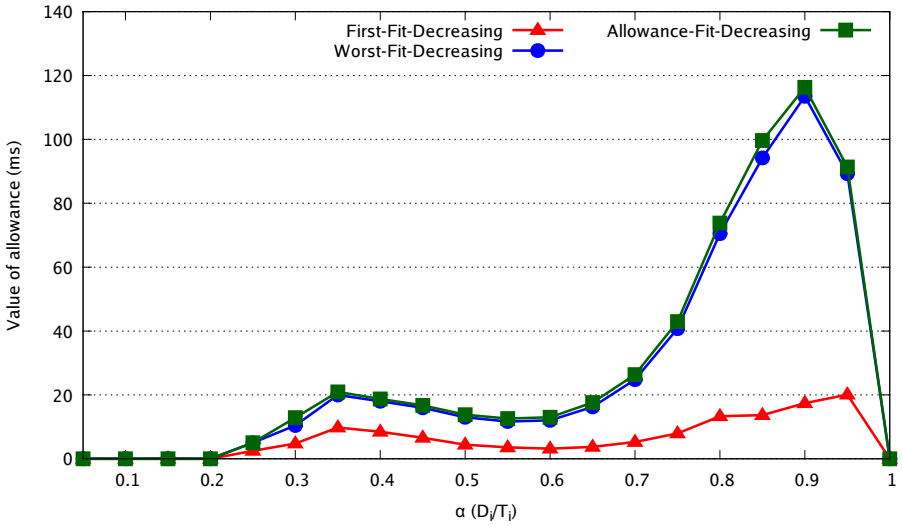

Fig. 3. Comparison of minimum *allowance* for the tasks partitioning heuristics on 4 processors.

We show in Fig.5 the comparison between the computational time of the three heuristics. AFD offers better results than WFD in terms of minimum *allowance*. But the computation time of AFD is 6 to 10 greater than the computation time of WFD. For a robust allocation to the WCETs overruns, it is interesting to use AFD. But when tasks must be accepted online, WFD is a preferable choice.

7 Conclusion

We have proposed a fixed-priority partitioned scheduling for homogeneous processors which maximizes the *allowance* of the execution duration. This scheduling is more robust than the others based on FF, BF or NF because during allocation of the tasks, the processor offering the greatest value of *allowance* is chosen. In terms of maximization of *allowance*, *Allowance-Fit* is slighty efficient than *Worst-Fit*. But in terms of computation time, *Worst-Fit* is largelly better than *Allowance-Fit*. Thus *Worst-Fit* is a good heuristic to maximize the robustness of a partitioned system of real-time tasks. In a future work, we will extend this approach to the class of *restricted migration* scheduling to improve the schedulability of our solution. In such a scheduling, the different jobs of a recurring task can migrate from a processor to another, but no migration is allowed during the execution of the job.

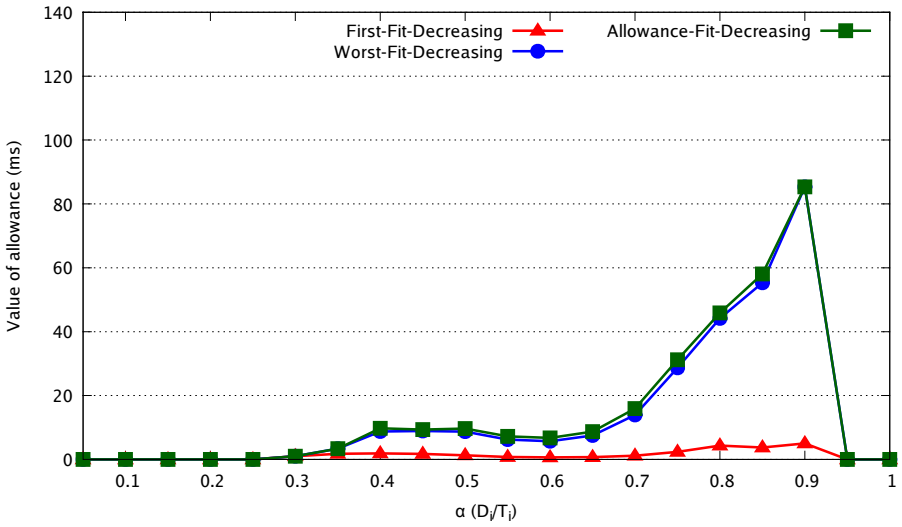

Fig. 4. Comparison of minimum *allowance* for the tasks partitioning heuristics on 8 processors.

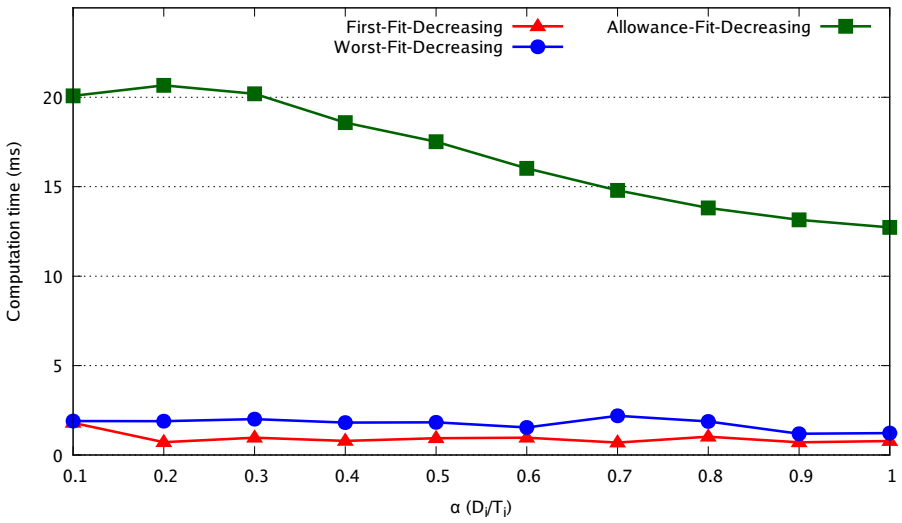

Fig. 5. Comparison of computation time of the heuristics.

References

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

2. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation **2**(4) (December 1982) 237–250
3. Davis, R.I., Burns, A.: Robust priority assignment for fixed priority real-time systems. In: Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS), Tucson, Arizona, USA, IEEE Computer Society (December 2007) 3–14
4. Bertogna, M.: Evaluation of existing schedulability tests for global EDF. In: Proceedings of the 38th International Conference on Parallel Processing Workshops (ICPPW), Vienna, Austria, IEEE Computer Society (September 2009) 11–18 First International Workshop on Real-time Systems on Multicore Platforms: Theory and Practice (XRTS).
5. Burchard, A., Liebeherr, J., Oh, Y., Son, S.H.: New strategies for assigning real-time tasks to multiprocessor systems. IEEE Transactions on Computers **44**(12) (December 1995) 1429–1442
6. Oh, Y., Son, S.H.: Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Systems **9**(3) (November 1995) 207–239
7. Andersson, B., Jonsson, J.: Preemptive multiprocessor scheduling anomalies. In: Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS), Fort Lauderdale, Florida, USA, IEEE Computer Society (April 2002) 12–19
8. Fisher, N.W., Baruah, S.K., Baker, T.P.: The partitioned scheduling of sporadic tasks according to static-priorities. In: Proceedings of the 18th Euromicro Conference on Real-time Systems (ECRTS), Dresden, Germany, IEEE Computer Society (July 2006) 118–127
9. Lakshmanan, K., Rajkumar, R., Lehoczky, J.P.: Partitioned fixed-priority preemptive scheduling for multi-core processors. In: Proceedings of the 21st Euromicro Conference on Real-time Systems (ECRTS), Dublin, Ireland, IEEE Computer Society (July 2009) 239–248
10. Dibble, P.: Jsr 1: Real-time specification for java (December 1998)
11. Bougueroua, L., George, L., Midonnet, S.: 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), Sainte-Luce, Martinique, IEEE Computer Society (April 2007) 8pp
12. Bini, E., Di Natale, M., Buttazzo, G.C.: Sensitivity analysis for fixed-priority real-time systems. In: Proceedings of the 18th Euromicro Conference on Real-time Systems (ECRTS), Dresden, Germany, IEEE Computer Society (April 2006) 13–22
13. Davis, R.I., Zabos, A., Burns, A.: Efficient exact schedulability tests for fixed priority real-time systems. IEEE Transactions on Computers **57**(9) (September 2008) 1261–1276
14. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. (1979)
15. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: A survey. In: Approximation Algorithms for NP-Hard Problems. PWS Publishing Co., Boston, MA, USA (1996) 46–93
16. Audsley, N.C., Alan, B., Tindell, K.W., Wellings, A.J.: Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal **8**(5) (Sepember 1993) 284–292
17. Bini, E., Buttazzo, G.C.: Biasing effects in schedulability measures. In: Proceedings of the 16th Euromicro Conference on Real-time Systems (ECRTS), Catania, Sicily, Italy, IEEE Computer Society (June - July 2004) 196–203