Integration of cache related preemption delay analysis into a priority assignment algorithm

Hai Nam Tran, Frank Singhoff, Stéphane Rubini, Jalil Boukhobza
Univ. Bretagne Occidentale, UMR 6285, Lab-STICC, F-29200 Brest, France
{hai-nam.tran,singhoff,rubini,boukhobza}@univ-brest.fr

ABSTRACT

Handling cache related preemption delay (CRPD) in preemptive scheduling context for real-time systems stays an open issue despite of its practical importance. Priority assignment algorithms and feasibility tests are usually based on the assumption that the preemption cost is negligible. Then, a system that could be schedulable on design time can fail to meet its timing constraints in practice due to preemption costs. In this article, we propose a simple approach to take into account the CRPD when performing priority assignment. The goal is to have a priority assignment algorithm which guarantees the schedulability of systems when tasks suffer CRPD on run-time. For such a purpose, we propose an extension of the feasibility test of Audsley and illustrate it with some examples. An implementation of our priority assignment method has been integrated to the Cheddar scheduling analyzer. A comparison of the proposed algorithm with classical priority assignment algorithms is achieved.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.2.4 [**Software Engineering**]: Software/Program Verification—*Validation*

Keywords

Real-Time Embedded System, Real-Time Scheduling, Priority Assignment

1. INTRODUCTION

Cache memories are crucial components to reduce the performance gap between processor and memory speed. The democratization of contemporary processors with large size and multi level caches in embedded real-time systems motivates the proposition of verification methods that are able to handle this hardware component.

One of the most important verifications performed for embedded real-time systems is schedulability analysis. Schedulability analysis provides some means to prove that an embedded system is *schedulable*. A system is said to be schedulable if all timing constraints of the tasks composing the software side of the system are satisfied. To study schedulability, many assumptions are usually made in order to simplify the analysis. One of them is that preemption cost is negligible. *Preemption cost* is the additional time to process interrupt, manipulate task queues and actually perform context switch.

Integrating a processor cache memory component in a embedded real-time system increases the whole performance in term of execution time, but unfortunately it may increase task preemption costs too. When a task is preempted, some memory blocks belonging to the task may be removed from the cache. Once the task resumes, the operating system needs to reload previously removed memory blocks into the cache. Thus, task preemption introduces a new preemption cost called the *cache related pre-emption delay* (CRPD), which is the additional time to refill the cache with cache blocks previously evicted on preemption time. To sum up, integrating cache in a embedded real-time system may increase execution time variability and decrease predictability of response times of a given task.

Problem statement. CRPD introduces two issues. First, it takes a high proportion in preemption cost and makes the preemption cost becoming not negligible anymore [12]. Second, the cumulative CRPD of a task set during a *feasibility interval* depends on priority assignment algorithm [20]. We recall that a feasibility interval is an interval for which testing of task's feasibility is needed [4]. If a system is not schedulable because of CRPD, we have to reorder the priority of tasks. However, there could be a possible outcome that the new priority assignment produces a bigger cumulative CRPD and still a non-schedulable system. Then, CRPD creates a cyclic dependency between scheduling analysis, priority assignment and preemption cost. To the best of our knowledge, there are no existing methods to assign priority to tasks and to guarantee that the system is schedulable while experimenting CRPD.

Contributions of this article. The contribution of our work is a method to perform priority assignment, which guarantees a system is schedulable while experimenting the impact of CRPD. To achieve this objective, we extended the feasi-

bility test proposed by Audsley [2] to be able to take into account the CRPD. The work is integrated into the Cheddar tool [16] which is an open source scheduling analyzer.

The rest of the paper is organized as follows. Section 2 presents the system models, notations and the gap between priority assignment and feasibility test caused by CRPD. Section 3 presents an overview of our approach to extend a feasibility test to take into account the impact of CRPD while performing priority assignment. In section 4, we explains a tree structure to deals with problems raised in section 3. In section 5, we discuss the complexity of the extended feasibility test. In addition, we compares the priority assignment produced by our approach with an example in [2]. Section 6 discusses related works. Finally, 7 concludes the article.

2. BACKGROUND AND NOTATIONS

In this section, we present the background and the notations required to introduce our priority assignment algorithm. The work in this article could be considered as an extension of the priority assignment algorithm and feasibility test of Audsley et al [2]. First, we present the system model and the notations. Second, we show an example to illustrate the limitations of classical fixed priority assignment algorithms. In addition, an overview of Audsley's optimal priority assignment (OPA) is given. At the end of the section, we discuss about feasibility test and CRPD analysis as well as point out our motivations.

2.1 System model and notations

We assume a single core processor system running n tasks with a preemptive fixed priority scheduler. Classical notations for real-time scheduling analysis are used [5] :

- $\tau_1, \tau_2, ..., \tau_i, ..., \tau_n$ are the tasks of the system.
- O_i, T_i, C_i, D_i are Offset, Period, Capacity and Deadline of task τ_i in the task set, respectively. Assuming system starts execution at time 0, a task τ_i makes an initial request at O_i, and then release periodically every T_i units of time. Each release of a task is called a *job*. It requires C_i units of time of computation and must finish before D_i units of time.
- $release(t_i^m)$ and $finish(t_i^m)$ are respectively the release time and the finish time of job m of task τ_i.
- $hp(i)$ (respectively $lp(i)$) is the set of tasks with priorities higher (respectively lower) than τ_i.

We extend these classical notations to model the CRPD when task τ_j preempts task τ_i. We note this delay by $crpd(\tau_j, \tau_i)$ in the sequel.

A job of a task can then be defined by a 3-uplets composed of the name of the task, the job release time and its capacity. For example, the 3-uplets $(\tau_i, 0, 1)$ refers to a job of task τ_i, which is released at time 0 and requires 1 unit of computation time.

Task	C	T	D	$priority_1$ **(RM)**	$priority_2$
A	1	4	4	1	2
B	2	7	7	2	3
C	3	7	7	3	1

Table 1: Task set example. The fourth column is priority assigned using RM. The fifth column is the priority assignment which makes the task set schedulable with CRPD

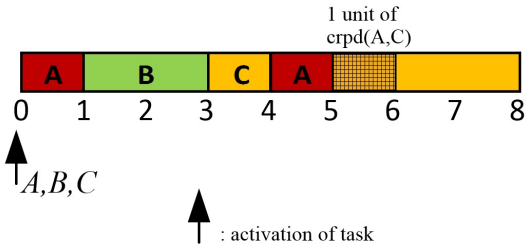

Figure 1: Scheduling simulation of the task set in Table 1 under Rate Monotonic priority assignment. Task C missed the deadline at 7 due to the preemption by task A.

2.2 Limitation of classical fixed priority assignment algorithm

With classical fixed priority assignment algorithm, priorities of all tasks are fixed before execution time. Classical schedulability methods do not considered CRPD in the system model. As a result, a system which has been verified as schedulable on design time may be unschedulable on execution time as CRPDs may occur. We give an example to illustrate this issue in Table 1. All tasks are released at time point 0.

With the task set in Table 1, we assume that $crpd(A, C) = 1$. We apply a *Rate Monotonic (RM)* priority assignment for which fixed priorities are assigned according to the period of the tasks. The lower period task has the higher priority level. This task set is not schedulable, as can be seen in Figure 1.

Only one priority assignment can make the task set schedulable: when we assign priority levels 2, 3 and 1 to respectively task A,B and C. This priority assignment can be reached by using the priority assignment algorithm proposed in this article.

2.3 Audsley's Optimal Priority Assignment

Let now introduce the Audsley's priority assignment algorithm. Audsley's priority assignment algorithm is said to be optimal in the sense that for a given system model, it provides a feasible priority ordering, resulting in a schedulable system, whenever such an ordering exists.

The pseudo code for the Audsley's algorithm is given below. For n tasks, the algorithm performs at most $n(n+1)/2$ schedulability tests and guarantees to find a schedulable priority assignment if one exists.

We have n priority levels corresponding to n tasks, with n is the lowest priority level. The algorithm starts by assigning the lowest priority level to a task. This task is called *assessing task* in the sequel. If the assessing task is not schedulable, the algorithm tries to assign the priority level to a different task, i.e. the assessing task is changed. If the assessing task is schedulable, the algorithm actually assigns this priority level to this task and then, moves to the next higher priority level. Then, it checks if a task in the unassigned priority tasks is feasible with this higher priority level. The algorithm continues until all tasks are assigned a priority level. If there are not any schedulable tasks at a given priority level, the system is not schedulable and the algorithm terminates.

```
for each priority level i, lowest first{
        for each unassigned task τ{
        if τ is schedulable at priority i{
                assign τ to priority i
                break (continue outer loop)
                }
        }
        return unschedulable
}
return schedulable
```

In the Audsley approach, a feasibility test is used to verify the schedulability of a task under a given priority level. We discuss about feasibility tests and CRPD analysis in the next section.

2.4 Feasibility test and CRPD analysis

With the system model presented in Section 2.1, for a given task set, the worst case response time R_i for each task τ_i can be computed and compared against the task deadline. The worst case response time occurs when all tasks are released at the same point in time (i.e. point in time called a critical instant). An iterative approach is used to compute the worst case response time of a given task. The task set is schedulable if every task meets their deadline (i.e. $\forall i : R_i \leq D_i$).

The equation to compute R_i is [8]:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \qquad (1)$$

To take into account the CRPD, the term $\gamma_{i,j}$ is introduced by [18]. $\gamma_{i,j}$ refers to the total cost of all pre-emptions due to jobs of task τ_j ($\tau_j \in hp(i)$) executing within the response time of task τ_i. $\gamma_{i,j}$ includes two delays. First, $\gamma_{i,j}$ includes the CRPD caused by jobs of higher priority tasks which preempt jobs of task τ_i. Second, $\gamma_{i,j}$ also includes the CRPD caused by jobs of higher priority tasks which preempt each other. In other words, $\gamma_{i,j}$ includes CRPD of task τ_j caused to the jobs of task τ_k, where τ_k has lower priority than τ_j, $\tau_k \in hp(i) \cap lp(j)$. The detailed computation of $\gamma_{i,j}$ is not in the scope of this article.

Then, the worst case response time of task τ_i can be computed by:

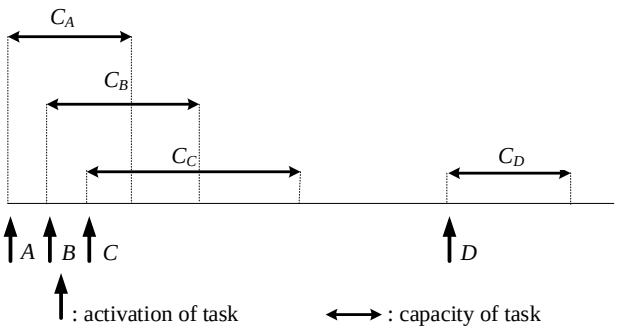

Figure 2: Four jobs released between the interval $[t, t + D_i)$ of job of task τ_i

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + \gamma_{i,j} \right) \qquad (2)$$

As we can see, the priority of tasks in the set $hp(i)$ must be known to identify the set $lp(j)$ and also $hp(i) \cap lp(j)$.

In [18], authors assumed that task priorities are preliminarily assigned when computing $\gamma_{i,j}$. In the Audsley's priority assignment algorithm, the assessing task is assumed to have the lowest priority so the set $hp(i)$ can be computed. Other tasks have higher priorities than the assessing task, however, relative priorities between those tasks are unknown. As a result, the set $lp(j)$ can not be computed and thus, it is not possible to compute $\gamma_{i,j}$. To perform priority assignment taking into account the impact of CRPD, we need a method to solve this issue.

3. OVERVIEW OF THE APPROACH

In this section, first, we give the basic ideas of our approach. Then, we point out some issues raised by our approach.

The idea of the approach is based on the feasibility test in [2]. Contrary to the approach of [18] which consists of computing the task worst case response time, we verify the feasibility for each job of those tasks during the feasibility interval.

A task is schedulable if its jobs released during the feasibility interval can meet their deadlines. Assuming a job of task τ_i is released at time t and has a deadline D_i, the job of task τ_i will experiment interference I_i caused by jobs of higher priority tasks during the interval $[t, t + D_i)$. Then, the job of task τ_i is feasible if:

$$C_i + I_i < D_i \qquad (3)$$

I_i may include three different delays. First, it may include the execution time of jobs of tasks in $hp(i)$ which are released during the interval $[t, t+D_i)$. Second, I_i may also include the CRPD caused by those jobs directly or indirectly preempting a job of task τ_i. Finally, I_i may include the CRPD caused by the those jobs preempt each other.

We focus on how to compute the upper-bound CRPD caused by those jobs of tasks in $hp(i)$ preempt each other, the third delay component. Figure 2 hows an example in which we assume that we have four jobs of higher priority tasks released between $[t, t + D_i)$. We also assume that the processor is idle before t. As priorities of tasks are unknown, computing cumulative CRPD is a problem.

One may wonder if task B preempts task A or not and would task C preempt task A or task B. We analyze those problems in sections 3.1 and 3.2. Because of the complexity of the problems, we chose to assess the CPRD of a set of jobs by creating a tree structure, which is presented in the next section.

In addition, these problems are also the reason we have to verify the feasibility of each releases of those tasks instead of computing the worst case response time.

3.1 Identifying relevant preemptions

The first problem is that with unknown task priorities, it is impossible to identify the preemptions that will occur. However, we can assess the jobs released during an interval and find the highest cumulative CRPD. The assessment is based on the job capacities, the release times and also CRPD between tasks.

A preemption may occur if the conditions of a *potential preemption* hold.

Definition 1. A potential preemption could be defined as the preemption between two jobs m and n of τ_i and τ_j, which could occur if $release(\tau_i^m) > release(\tau_j^n)$ and $finish(\tau_j^n) > release(\tau_j^m)$. Those conditions are defined based on the work in [7].

We also have to consider the case where a task is preempted several times. As we can see in the figure 2, if task A has the lowest priority, it will be preempted by task B and then later by task D.

It is also important to notice that we should take into account the CRPD caused by indirect preemptions. For example, if task B preempts A and then C preempts B, then C can remove useful data of A in the cache.

Finaly, not every preemption causes CRPD. Thus, the highest number of preemptions could not correspond to the highest value of cumulative CRPD.

3.2 Implicit priority policy

A second issue is related to priority. When we have several tasks which were preempted and which are waiting for the processor, there could be an *implicit priority policy* was set.

We illustrate this problem by an example. If task B can preempt A, it means that the priority of task B is higher than A. If task C preempts B, then, at the time that B finishes, B should be executed first. In case later we have an activation of task D, D could preempt B. In other words, if B preempted A, D will not preempt A.

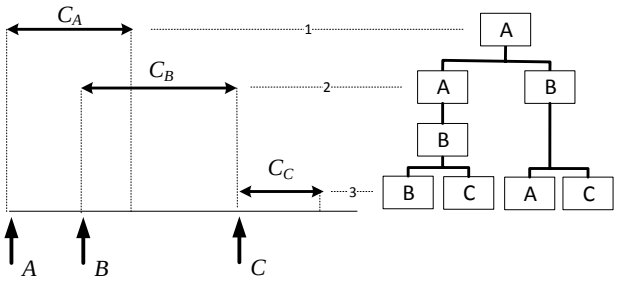

Figure 3: An example of computing the tree with 3 jobs. Only the decisions of the scheduler to choose which task to run are shown.

In conclusion, we have two problems to compute the maximum CRPD of a set of task releases when task priorities are unknown. In the next sections, we propose a mean to solve them by building a tree structure.

4. CRPD TREE

In this section, we present our method to deal with problems listed above in order to compute the third delay component of I_i. First, we introduce the CRPD tree, which is a tool used to assess the CRPD of a set of jobs. Second, we show how to compute the tree and provide an example to demonstrate the method.

For a set of jobs with unknown task priorities, we use a tree structure to compute all preemption schemes. Second, the conditions of relative priority between tasks, which is related to a specific cumulative CRPD value. The result of the computation includes two parts. The first part is the possible cumulative CRPDs for this set. The second part is the conditions of relative priority between tasks, which is related to a specific cumulative CRPD value.

The tree $T = (N, E)$ is defined by N, the set of node and E, the set of edge. Each node n of N models a running job at a point of time. Each node n is defined by a 4-uplet $(a, b, c, d,)$ where a is a point in time, b a task name, c the remaining capacity of the task and d its CRPD. Each edge e from E models a decision of the scheduler. We can have three types of edges modeling three different events during scheduling: allowing preemption, not allowing preemption, choosing a job in the set of jobs waiting to be run.

4.1 How to compute the tree

In this section, we present how the tree can be built from a set jobs of tasks. The tree is computed by using a recursive algorithm. The main idea is to assess all higher priority jobs released between the time interval $[t, t + D_i)$ of job released at t of task τ_i. Jobs are ordered according to their release times. We start from the first job. At the release time of the job, we check if there are any previous jobs which are running or which are ready to run.

- If there are no jobs, we add a node to the tree, to mark the activation of the current job.
- If there is a job running, we add 2 nodes. One node

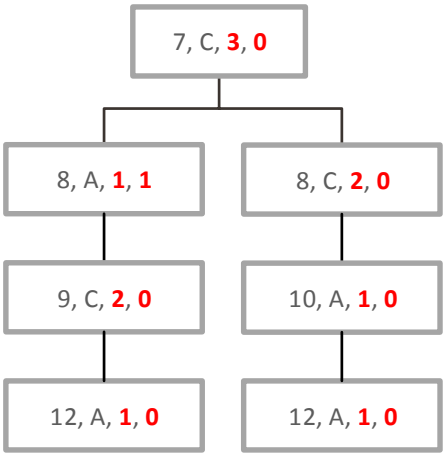

Figure 4:

represents the preemption occurred and one node represents the preemption did not occur, as can be seen in Figure 3 when assessing the activation of job of task B (second job). By doing that, we can analyse all possible preemption decisions of the scheduler, which is the problem presented in section 3.1.

- If there are q jobs ready to run, we have to add $q+1$ nodes with one additional node presents the current checking job. The new nodes represents the different decisions of the scheduler to choose a job to run, as can be seen when assessing the activation of job of task C (third job) in the right branch of Figure 3. When the job of task C is activated, the scheduler has to choose to run task A, which was prempted by B, or task C. In addition, it is also important to take the relative priority problem in section 3.2 into account when we have this case.

Then, we move on the next job. The algorithm finishes when all jobs are checked.

We also see that in the left branch of Figure 3, there is an extra node of job of task B added. It is because this job executes when job of task A is finished but job of task C is not released.

4.2 Example

In this section, we present an example to illustrate the algorithm. We use the task set of Table 1. We test if task B is feasible with priority level 3. The first release of B at time point 0 is feasible, we skip the presentation of this test. We check the second release of B at time point 7.

In the duration $[7, 7+D_B)$, there are three releases: $(C, 7, 2)$, $(A, 8, 1)$ and $(A, 12, 1)$. The assumption about CRPD is that $crpd(A, C) = 1$. The interference caused by the capacity of 3 jobs is 5 units of time.

Figure 4 is the output of the algorithm. If the release of C at 7 is preempted by release of A at 8, there is 1 unit of CRPD

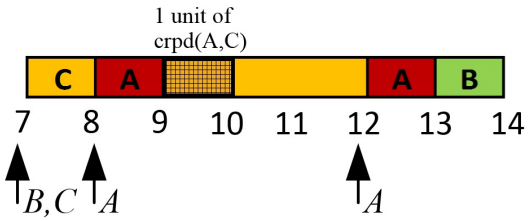

Figure 5: Scheduling simulation of left branch

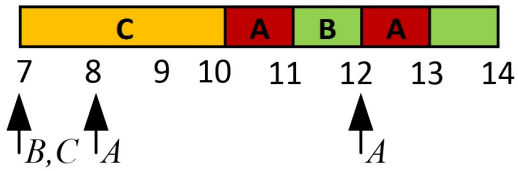

Figure 6: Scheduling simulation of right branch

added to the capacity of C, as can be seen in Figure 5. If not, we have the scheduling simulation as seen in Figure 6.

$if\ A\ preempts\ C => I_B^7 = C_B + 5 = 7 = D_B$
$if\ A\ doesnot\ preempt\ C => I_B^7 = C_B + 5 + 1 = 8 > D_B$

From this analysis, we can see that task B can be feasible at priority level 3 if the task A does not preempt task C. It means that A has lower priority than C. When continuing the test, we see that both A and C cannot be feasible at priority level 3 and B is feasible with a condition. As mentioned in section 3, the task set is feasible if we assign task A, B, C priority level 2, 3 and 1, respectively.

5. EVALUATION

To evaluation our algorithm, we first analyze its complexity and then, we compare its results to a classical priority assignment algorithm such as the Audsley algorithm.

5.1 Complexity of the algorithm

The complexity of Audsley's optimal priority assignment algorithm is $O((n^2+n) \cdot E)$, where E is the complexity of the feasibility test.

In our algorithm, for each feasibility test we have to create one CRPD tree with a complexity about F. Thus, the complexity becomes $O(((n^2+n) \cdot E) \cdot F)$

To compute F, the complexity of the tree, we must compute the number of nodes in the tree. The number of nodes in the tree depends on how many potential preemptions and implicit priority occur (see section 3.1 and 3.2). Indeed, we have one potential preemption, we need to add two new nodes.

In the worst case, build the tree has a complexity similar to the complexity of performing scheduling simulation with all possible priority assignment schemes. Then If we have n priority levels and m jobs released during an interval, the worst case number of node is about $n! * m$.

Task	C_i	T_i	D_i	O_i	Audsley	According to CRPD
A	1	10	1	4	1	2
B	1	10	2	5	4	4
C	5	20	6	0	2	1
D	8	40	9	7	3	3
E	8	40	14	27	6	6
F	6	40	30	0	5	5

Table 2: Comparing our algorithm to the Audsley priority assignment algorithm

5.2 Applying our algorithm to the Audsley's case study

To evaluate the priority assignment algorithm with updated feasibility test, we compare its results with the Audsley's algorithm. The task set example in table 2 has been proposed in [2] to access the efficiency of the Audsley algorithm.

We apply our priority assignment algorithm on this case study with the following assumption on CRPD: $crpd(A, C) = 1$, $crpd(B, D) = 1$, $crpd(C, E) = 1$ and $crpd(B, F) = 1$.

In the Table 2, the column 5 shows priority assigned by Audsley algorithm and the column 6 the priorities assigned by the algorithm proposed in this article.

From this case study, we can see that the algorithm with our updated feasibility test lowering the priority of task A because of the assumption $crpd(A, C) = 1$. This assumption makes task C not schedulable at priority level 2. Thus, task set is not schedulable with the the original priority assignment while is it schedulable with ours.

6. RELATED WORKS

In this section, first, we present related works to priority assignment algorithms. Second, we discuss about the computation of CRPD and its integration in scheduling analysis.

One of the most known priority assignment result is the Liu and Layland [13] one which showed that for synchronous periodic tasks with deadlines on requests *Rate Monotonic* is the optimal priority assignment algorithm.

Latter, assumptions on task release times and deadlines have been relaxed. For example, Leung and Whitehead [11] showed that for synchronous tasks with deadlines less than or equal to their periods *Deadline Monotonic* (DM) is optimal instead. Furthermore, audsley [2] addressed asynchronous periodic tasks with arbitrary deadlines.

Davis and Burns [6] improved Audsley's algorithm by taking tolerable interference into account. The algorithm assigns priority to a task, which is not only feasible but also can tolerate highest number of interference.

Finally, Wang et al [21] proposed preemption threshold to improve the schedulability and to reduce preemption overhead. However, in their models, the preemption cost is assumed to be zero.

Most of the research in priority assignment algorithms on single-processor systems have mainly focused on finding the optimal priority assignment algorithm for specific system models. To the best of our knowledge, there is no research in the domain of priority assignment taking CRPD into account.

However, with the arrival of multiprocessor architectures in embedded real-time systems, many works have been made on CRPD computation. The computation of CRPD, when a preemption occurs, is based on the two classical analysis methods called Useful Cache Blocks [10] and Evicting Cache Blocks [3]. Both of them have lead to many extensions [10, 3, 19, 1].

Then, the integration of CRPD into fixed priority scheduling analysis can be made by adding CRPD into the worst case response time equation [10, 3, 17],

Scheduling analysis with CRPD and dynamic scheduling has also been investigated. Ju et al.[9] and Luniss et al.[15] developed CRPD analysis for pre-emptive Earliest Deadline First (EDF) scheduling.

In addition, feasibility tests, which took CRPD into account or not, obviously assumed that priorities of tasks are known, which is not the case in this article.

7. CONCLUSIONS

In this article, we presented an approach to perform priority assignment with CRPD. The problem we are dealing with is the lack of feasibility tests taking into account the impact of CRPD, in order to be applied during priority assignment.

We used Audsley's optimal priority assignment algorithm and extended the original feasibility test proposed in [2] to consider the impact of CRPD. The approach consists in verifying the feasibility of each job of each task during its feasibility interval while accounting the interference due to jobs of higher priority tasks. CRPD is a part of this interference. The algorithm to compute the CRPD due to by jobs of tasks with unknown relative priorities handles a tree structure storing potential preemptions. We have evaluated this algorithm by comparing its performances with classical priority assignment algorithms.

There are many open problems that we could address in the future. First, we have not investigated the optimality of the method. Second, the complexity of the method has to be precisely characterized. Finally, we plan to investigate how to combine the feasibility test with the memory layout optimization as suggested in [14].

8. REFERENCES

[1] S. Altmeyer, R. I. Davis, and C. Maiza. Pre-emption cost aware response time analysis for fixed priority pre-emptive systems. *32nd RTSS*, 2011.

[2] N. C. Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer, 1991.

[3] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Real-Time Technology and Applications*

Symposium, 1996. Proceedings., 1996 IEEE, pages 204–212. IEEE, 1996.

[4] L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*, pages 397–404. IEEE, 2006.

[5] R. I. Davis. Burns standard notation for real-time scheduling. In *Proceedings of the conference organized in celebration of Professor Alan Burns sixtieth birthday*, page 38.

[6] R. I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 3–14. IEEE, 2007.

[7] R. Dobrin and G. Fohler. Reducing the number of preemptions in fixed priority scheduling. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 144–152. IEEE, 2004.

[8] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[9] L. Ju, S. Chakraborty, and A. Roychoudhury. Accounting for cache-related preemption delay in dynamic priority schedulability analysis. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pages 1–6. IEEE, 2007.

[10] C.-G. Lee, H. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *Computers, IEEE Transactions on*, 47(6):700–713, 1998.

[11] 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, 1982.

[12] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. In *Proceedings of the 2007 workshop on Experimental computer science*, page 2. ACM, 2007.

[13] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[14] W. Lunniss, S. Altmeyer, and R. I. Davis. Optimising task layout to increase schedulability via reduced cache related pre-emption delays. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 161–170. ACM, 2012.

[15] W. Lunniss, S. Altmeyer, C. Maiza, and R. I. Davis. Integrating cache related pre-emption delay analysis into edf scheduling. *University of York, York, UK, Technical Report YCS-2012-478. Available from http://www-users. cs. york. ac. uk/˜ wlunniss*, 2012.

[16] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.

[17] J. Staschulat and R. Ernst. Scalable precision cache analysis for preemptive scheduling. In *ACM SIGPLAN Notices*, volume 40, pages 157–165. ACM, 2005.

[18] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 41–48. IEEE, 2005.

[19] Y. Tan and V. Mooney. Timing analysis for preemptive multitasking real-time systems with caches. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(1):7, 2007.

[20] H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza. Instruction cache in hard real-time systems: modeling and integration in scheduling analysis tools with aadl. In *The 12th IEEE International Conference on Embedded and Ubiquitous Computing*. IEEE, 2014.

[21] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Real-Time Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference on*, pages 328–335. IEEE, 1999.