Integration of Cache Related Preemption Delay Analysis in 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 subject 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 an elementary approach to take into consideration the CRPD while assigning priorities to tasks. The goal is to acquire a priority assignment algorithm guaranteeing the schedulability of system when tasks suffer CRPD at run-time.

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. PROBLEM STATEMENT

Cache related preemption delay (CRPD) introduces two issues. First, it accounts a high proportion in preemption cost and causes the preemption cost to be significant [5]. Second, the cumulative CRPD of a task set during its *feasibility interval*, which is an interval for which testing of task feasibility is needed [2], depends on the applied priority assignment policies.

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 [6], [4], [1], [3]. To the best of our knowledge, there are no existing work in the domain of priority assignment taking CRPD into account.

If a system is not schedulable under a priority assignment policy because of CRPD, a method to work around could be reordering task priorities. The goal is to reduce the total CRPD while still maintaining the feasibility of tasks. For example by lowering the number of preemptions. The solution consists in evaluating CRPD of tasks produced (by timing analysis),the total number of preemptions and also the feasibility of each tasks. As far as we know there are no existing method to achieve this goal. The problem is that CRPD depends on applied priority assignment policies and solving this dependency is still an open issue.

2. INTEGRATION OF CRPD ANALYSIS IN PRIORITY ASSIGNMENT ALGORITHM

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. Assuming system starts execution at time 0, a task τ_i makes an initial request at offset O_i, and then releases 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. Exact computation of CRPD for each pair of tasks, for example when task τ_A preempts task τ_B, using timing analysis technique is out of the scope of this article.

To break the dependency stated in Section 1, we propose a priority assignment algorithm, if it finishes assigning a priority level to each tasks of the system, the resulted system is guaranteed to be schedulable while experimenting the impact of CRPD. It includes the integration of a feasibility test into the priority assignment algorithm of Audsley [1]. For n tasks, Audsley's 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 being 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 whether 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. The pseudo code for the Audsley's algorithm is given below.

```
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 original work of Audsley, that does not take CRPD into account, a feasibility test is used to verify the schedulability of a task under a given lowest priority level while relative priorities among higher priority tasks are unknown. Existing feasibility test with CRPD taken into account is not able to test the feasibility of a task in this context. For example, in [7], the authors assumed that task priorities are preliminarily assigned.

Contrary to the approach of existing feasibility tests which consists of computing a task worst case response time, we verify the feasibility for each job of a task during its feasibility interval. A task is schedulable if all its jobs released during the feasibility interval can meet their deadlines. Assuming a job of task τ_i is released at time t, requires C_i unit of computation time and must finish before D_i, the job experiences interference I_i caused by other jobs of higher priority tasks during the interval $[t, t + D_i)$. Then, the job of task τ_i is feasible if:

$$C_i + I_i < D_i \quad (1)$$

I_i includes three delays components. First, it is the execution time of jobs of higher priority tasks which are released during the interval $[t, t+D_i)$. Second, I_i includes the CRPD caused by those jobs of higher priority tasks directly or indirectly preempting a job of task τ_i. Third, I_i also includes the CRPD caused by the those jobs preempt each other.

Computation of the third component of I_i is the main challenge when tasks priorities are not assigned. A naive approach is to perform scheduling simulation for those jobs with all possible combinations of priority assignments and compute the cumulative CRPD.

We propose a solution to compute that third delay component based on three information: *release time*, *capacity* and *CRPD relationship*. The idea is to compute a tree structure, which presents all preempting decisions the scheduler has to do. 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 an instant, b a task name, c the remaining capacity of the task and d the CRPD caused by the activation of this task. 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. Each branch of the tree presents a non-conflict priority assignment scheme. By non-conflict, we mean that there are not any cases two tasks can preempt each other, for example, if τ_A preempts τ_B then later τ_B cannot preempt τ_A. Based on the tree, we can compute the upper-bound cumulative CPRD.

We illustrate the tree by an example in Figure 1. Assuming that we have information of each job activation time, capacity and the CRPD when those tasks preempting each other. In this example, we have the activation of jobs of three task A, B and C in an interval. By computing the tree, we can assess all possible preemptions of the jobs, which can occur in a specific priority assignment scheme. Thus, evaluating the cumulative CRPD in each branch of the tree.

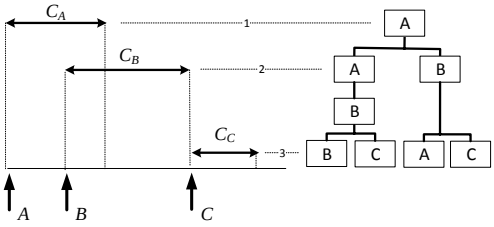

Figure 1: An example of computing the tree with 3 jobs. Only the decisions of the scheduler to choose which task to run are shown.

3. CONCLUSIONS

In this article, we discuss a problem of CRPD and priority assignment and an elementary approach to solve this issue. We extend Audsley's optimal priority assignment algorithm [1] by evaluating 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, including CRPD. A tree structured is implemented to compute the CRPD of jobs with unknown task priorities during an interval. We plan to evaluate the proposed approach in terms of complexity and optimality in the future works.

4. REFERENCES

[1] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. In *Technical Report YCS 164, Dept. Computer Science*. University of York, UK, 1991.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.