IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

Runtime CRPD Management for Rate-Based Scheduling

José Marinho

Stefan M. Petters

HURRAY-TR-101201

Version:

Date: 12-04-2010

Runtime CRPD Management for Rate-Based Scheduling

José Marinho, Stefan M. Petters

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Temporal isolation is an increasingly relevant concern in particular for ARINC-351 and virtualisation-based systems. Traditional approaches like the rate-based scheduling framework RBED do not take into account the impact of preemptions in terms of loss of working set in the acceleration hardware (e.g. caches). While some improvements have been suggested in the literature, they are overly heavy in the presence of small high-priority tasks such as interrupt service routines. Within this paper we propose an approach enabling adaptive assessment of this preemption delay in a temporal isolation framework with special consideration of capabilities and limitations of the approach.

Runtime CRPD Management for Rate-Based Scheduling [*]

José Marinho Stefan M. Petters
CISTER, ISEP
Portugal
{*jmsm,smp*}*@isep.ipp.pt*

Abstract

Temporal isolation is an increasingly relevant concern in particular for ARINC-351 and virtualisation-based systems. Traditional approaches like the rate-based scheduling framework RBED do not take into account the impact of preemptions in terms of loss of working set in the acceleration hardware (e.g. caches). While some improvements have been suggested in the literature, they are overly heavy in the presence of small high-priority tasks such as interrupt service routines. Within this paper we propose an approach enabling adaptive assessment of this preemption delay in a temporal isolation framework with special consideration of capabilities and limitations of the approach.

1. Introduction

Timely execution is of utmost importance in hard real-time systems. Many systems containing hard real-time elements will also feature parts of soft real-time or best-effort character. The nature of the latter tasks is that they are subject to at least occasional *misbehaviour*, in contrast with hard real-time tasks. The reasons for such misbehaviour may be, for example, deliberate or accidental underestimation of their worst-case timing requirements so as to increase overall system efficiency. In order to guarantee that this misbehaviour does not affect hard real-time requirements, resource management mechanisms need to be integrated into the system. These mechanisms create a barrier that ensures temporal isolation under the condition that tasks do not communicate or share resources beyond the processor.

Basic earliest-deadline first (EDF) scheduling does not enforce temporal isolation at all. In the case of independent tasks whose deadlines are equal to their periods, a task set is schedulable in EDF if the sum of the task utilisations is at most unity [1]. The previously described mix of applications might lead to tasks executing for more than they have been accounted for in the analysis. In this situation overall system utilisation may cross the limit and deadlines may start to be missed. This shortcoming has been addressed by, for example, constant bandwidth servers (CBS) [2] and RBED[3] frameworks for simple task models. Both approaches have added budget enforcement on top of EDF.

Modern microprocessors suitable for deploying aforementioned temporal isolation frameworks have, usually, caches to reduce the memory to processor communication bottleneck. Although this enhances system performance, there is inherent non-determinism (from the perspective of the programmer or system integrator) in a multi-programming environment due to the concurrent access to this resource. One task preempting another will not only create direct delay, by executing instead of the preempted task, but also indirect impact by replacing useful cache contents of the preempted task with content of its own. This is generally termed *cache-related preemption delay* (CRPD). Besides caches, other acceleration techniques also cause additional delays, because of loss of an existing working-set on preemption. Examples of this are dynamic branch prediction or TLB entries. Within this paper we will concentrate discussion on CRPD which we will use as synonymous for a bound on the CRPD.

Strict temporal isolation is no longer ensured in RBED when CRPD is added to the task model if this is not accounted for in the worst-case execution time (WCET) [4] computation. Solutions for CRPD estimation are available in literature, but these rely only on static analysis of code [5], which makes usage in an open system impractical [6, 7]. Microprocessor usage may be enhanced if on-line mechanisms are used to tune off-line estimations, when these are available. We envisage in this work the use of hardware found in a number of processors, allowing certain system events like cache misses, translation look-aside buffer misses,

[*]This work was supported by the Portuguese Science and Technology Foundation (FCT) (CISTER FCT-608) and ARTEMIS-JU (RECOMP project ARTEMIS/0202/2009).

branch predictions and so forth to be counted. Such performance monitoring counters are available, for example, in current x86, some power PC and some ARM processors.

In this work we are assuming that tasks within a system will have different timing requirements, but can be grouped in distinct categories. Those are: hard real-time, soft real-time and best effort tasks. The soft real-time tag describes tasks that may occasionally incur in deadline violations, whereas best effort tasks will have no real-time requirements. Best effort tasks need, nevertheless, to be considered in the framework to ensure fairness and progress (i.e. this tasksshould receive some share of the processor even when the system is heavily loaded). The cache is either assumed to be directly mapped, in which case the problem has an easy solution, or set-associative with a well-behaved replacement policy, thus invalidating the occurrence of a stream of cache misses following a preemption. In the computation of a value for CRPD, we assume that the system only has one cache level which is physical address-tagged in order to get a constant time value for a cache miss. This restriction will be relaxed in the future to accommodate more cache levels, but, as of this introductory presentation, this simple model is adopted. In the devised system every task has to pass an acceptance test prior to its first job execution.

Past work considered the preemption delay as a constant value for all tasks[5]. This has serious drawbacks when small tasks, like interrupt service routines are considered.

The present document provides a description of a run-time and on-line mechanism where temporal isolation is provided for hard real-time tasks in an open system environment, in which new tasks may dynamically be added or removed. The architecture envisioned is not restricted to on-line usage, may also be used in an off-line analysis framework to tune the estimations across operation.

In the next section we will provide an overview of related work. In Section 3 we present the system model and we will also give a very brief review of the RBED framework, which will be used as a point in case during the discussion. Considered possible preemption scenarios and the general logic of the run-time mechanisms are described in the next section. Section 5 is dedicated to the generalisation of the proposed approach to frameworks other than RBED and different cache architectures other than direct-mapped. Conclusions and indication of future work finalises the document.

2. Related Work

CRPD has been a subject of wide study. Several methods have been proposed that provide an off-line estimation based on static analysis, for this inter-task interference value. In this context the term *off-line* implies that the analysis is preformed prior to system deployment or integration, while *static* implies that the analysis is performed using analytical models rather than measurements.

Lee et al. presented one of the earliest works on CRPD estimation for instruction-caches [6]. The authors introduced the concept of useful cache blocks which describe cache blocks that will potentially be reused in the later program execution. They assume that the CRPD incurred by a task, during preemption by another task, is constant throughout a basic block. This is a valid assumption for instruction caches, but does not hold for data caches. With both concepts they devised an algorithm that estimates the number of cache blocks of each task and compares this data with the used cache blocks of possibly preempting tasks.

This work was later improved by the same group [7] to eliminate infeasible preemptions, thus improving the CRPD estimate. The CRPD estimation in these papers is formulated as an integer linear programming problem. Building on this work, Ju et al. [8] increased the accuracy of the CRPD estimation by using a more extensive data structure in the assessment of useful cache blocks. They have also moved the approach into a dynamic priority assignment (i.e. EDF).

Computation of the CRPD in data caches has been proposed by Ramaprasad and Mueller [9]. Since the assumption used in [6], that the value of CRPD throughout a control flow graph's basic block would remain constant, no longer holds for data caches a different approach had to be devised.

Ramaprasad and Mueller integrated the computation of data-cache related preemption delay in their previous framework for WCET computation. Code is statically analysed in order for a cache hit/miss pattern to be computed. In this way useful cache blocks are computed. The CRPD is then computed for every program point. This cost is related to the number of useful cache blocks that can be evicted by higher priority tasks. Subsequently, the n highest preemption costs are selected correspondingly, where n is the maximum number of preemptions iteratively determined considering assigned priorities and WCET of the tasks.

Later, an improved method was proposed by Ramaprasad and Mueller [10] that tightens the number of preemptions in relation to the work of Lee et al. [6]. They also added further considerations on the possible preemption points. This is achieved by considering BCET together with the WCET. The improvement relies on phasing considerations in the hyperperiod of the task-set, which becomes problematic in terms of analysis time required when the task-set is not hyperperiodfriendly.

Embedded in all the stated frameworks are schedulability tests. Scheduling analysis for [6] is based on response time analysis (RTA); [8] provides a demand bound function based procedure, while the general approach of computing the CRPD is similar.

A different approach relevant to our work is the one followed by Buttazzo et al. [11]. In order to reduce CRPD, the usage of non-preemptive areas of code is proposed. The preemption points are thus reduced to a small number of well defined points. In this way the maximum CRPD is decreased and overall system's response time is enhanced. This may in turn render a task-set schedulable that would not be so without considering CRPD overhead.

A different approach is taken by Altmeyer and Gebhard [12]. They have proposed minimizing the amount of cache lines evicted, by changing the memory accessed by the program. An interesting different angle in the context of a conservative static WCET analysis approach is taken by Altmeyer and Burguiere [13]. As the static analysis approach followed only assumes cache hits, when it can be proven that a certain memory reference is in the cache, the CRPD calculation only needs to account for those references known to be in the cache. This exploits the inherent pessimism of static analysis tools in the presence of many cache accesses which are classified as potential but not definite hits.

All the cited approaches so far can not be applied in an open system architecture as they rely on computationaly heavy static analysis of all code in the system, which is not amenable to on-line use. Our approach differs in the sense that it provides the foundation to deal with on-line estimated CRPD.

3. System Model

Within this work we focus on the temporal isolation framework RBED with some extensions. We will briefly introduce the concepts related to our work, but direct the interested reader to the original work [3, 14, 5] for further details. At the end of the section we provide in Table 1 an overview of the used terminology for reference.

An EDF scheduler forms the core of RBED. However, the scheduled entities are in first instance budgets which have certain tasks attached to them, rather than the tasks themselves. For ease of presentation no difference is made between the budgets and their associated tasks in the majority of the discussion. We will highlight the difference where relevant.

The budgets are replenished according to the rules used as assumptions in the schedulability test. The original work [3] used an utilisation based test under independence assumption of the scheduled tasks, while later work [5] adds a schedulability analysis based on a demand-bound function to the framework. Budgets are treated as tasks for analysis purposes in both approaches. A task τ_i is associated with a budget E_i. Furthermore the budget has a relative deadline after release D_i (which is used for scheduling decisions) and a function which describes the release of budgets. In the simplest case, which has been used in the original work, the budgets are released periodically with an inter-arrival time of T_i and a deadline of $D_i = T_i$. The later work allowed for alternative arrival patterns for budget releases, as well as arbitrary deadlines. Again, for ease of presentation, we will use throughout the discussion the strictly periodic model with implicit deadlines, however, we will also show that the proposed solutions apply to the more advanced task model.

Temporal isolation is achieved by enforcement of the budgets, which means that once a budget is exhausted, a reschedule is initiated. As previously stated, replenishment of exhausted budgets follows the assumptions made in the analysis. Again working on the simplest case, a budget release coincides with a task release; i.e. one instance of the budget E_i is used to execute one job $J_{i,n}$ of task τ_i. Only budgets which have a task which is *ready* associated with them may be scheduled.

One important aspect of budget-based scheduling is the fact that in most cases the budget will not be exhausted thus creating slack in the system; i.e. execution time reserved, but not used. Lin and Brandt [14] have introduced various slack management approaches. The most relevant aspect of the slack management is that, under the assumption that slack is passed to a task which is ready, the deadline used for scheduling this slack may be relaxed to an arbitrary point in the future without jeopardising the guarantees provided by the analysis.

Another important concept from the work of Brandt et al. [3, 14] is that we distinguish two relevant issues following from the slack management described. Firstly, slack is strictly required to be passed to a runnable task when a task becomes idle. Secondly, a task may use future budgets when its current budget is exhausted, subject to the task being scheduled with the credentials (i.e. deadline) of the future budget.

RBED also introduced an admission and resource management entity call Resource Allocator (RA). The RA is responsible for assigning budgets to task. In an open environment every newly arriving task has to provide its requirements to the resource allocator. In the simple model, this is restricted to WCET C_i and minimum-inter arrival time T_i which is also the implicit deadline D_i. On arrival and departure, the RA performs not only the allocation of budgets to tasks,

C_i	WCET of a given task τ_i
T_i	period/minimal inter-arrival time of task τ_i
D_i	relative deadline of task τ_i
E_i	budget allocated to task τ_i for one job
y_i	worst-case number of cache misses additionally induced by task τ_i on another task during preemption
Y_i	worst-case increase of execution time caused by the additional y_i cache misses in the tasks preempted by τ_i.
Z_i	execution time of a job of task τ_i observed

Table 1. Terminology Used

but also the schedulability analysis.

During execution a task will displace a number of cache entries. The amount of *damage* a task τ_i causes in a preempted task τ_j, (CRPD) is dependent on the cache architecture and replacement scheme. In a direct-mapped cache the cache footprint equals the worst-case amount of damage denoted by y_i, while in n-way set-associative caches with least-recently used or FIFO replacement algorithms, a single access to one element of the set may cause n additional cache misses in the preempted task. Again, for ease of presentation we assume a single level direct mapped cache for the discussion, bearing in mind that other cache-architectures are possible with only minor changes to the equations. The additional y_i cache misses caused in the preempted task give rise to an execution time increase of Y_i. At this stage there is a linear relation between y_i and Y_i but this can be easily changed to adapt to multi-level memory architectures and other sources of preemption delay, like branch prediction or translation look-aside buffers. In the discussion, we will however limit ourselves to CRPD. In this context we also assume that the impact of a preemption is bounded. This rules out some cache-replacement algorithms, like random cache replacement.

We assume that the values C_i, T_i and Y_i presented above are readily available within the system, albeit might be in the form of conservative estimates, like the WCET. The most conservative estimate for Y_i is that a preemption removes the entire useful cache content in the system, which means that entire cache replacement has to be considered. During execution of a single job of task τ_j we assume that the system is able to observe the actual execution time of the job (Z_j) ignoring the preempted times but not the preemption delay.

4. Preemption Scenarios

In the first two subsections it is assumed that a single job will behave properly and execute within its budget when running without being preempted. Furthermore we assume that out of any preemption delay budget (Y_i) it receives as compensation for preemptions, only at most the deserved part of that preemption delay compensation will be used and remainder is passed. The assumption is then relaxed in Section 4.3

4.1. Simple Preemption

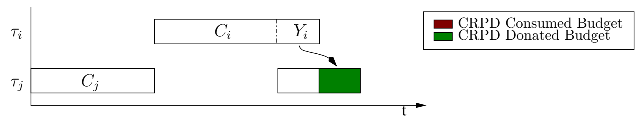

Figure 1. Simple Preemption

Besides the budget assigned to each task, there is a supplementary budget value (Y) that accounts for the possible cache interference caused to jobs of other tasks. This additional value is added to the donating task's budget when performing the schedulability test. The simplest preemption scenario is a system of two tasks where τ_j is preempted by τ_i. This is depicted in Figure 1. In this case τ_i would pass Y_i to τ_j. The available budget for τ_j execution would thus be increased by Y_i units of time. The value assigned to Y may be variable, evolving with system execution. The initial approximation could be such that Y would be equal to the time needed to refill the entire cache, since this is guaranteed to be the worst case. Subsequently this value may be reduced to a more realistic scenario.

When task τ_i finishes its execution, τ_j will obtain access to the processor executing on its budget plus the added Y_i. Execution of τ_j's current job will eventually come to a halt. The remaining value is handled as slack. Because of the assumptions made, the system will be idle and thus it needs to be handled as described by Petters et al. [5].

The CRPD budget passing in this situation is similar to slack donation. It is essentially the same idea as principle 2 in Lin and Brandt's work [14], since Y can be perceived as slack resulting from overallocation of budget by the preempting task that is then handed over to the preempted task. Due to this fact, it is equally valid in terms of maintaining the premisses used during schedulability analysis. The budget passed in this form will be executed with a longer relative deadline, since under EDF a task can only preempt running tasks with longer relative deadlines. This will always constitute a relaxation of the local processor demand. As in slack donation, no budget can be passed to a blocked task.

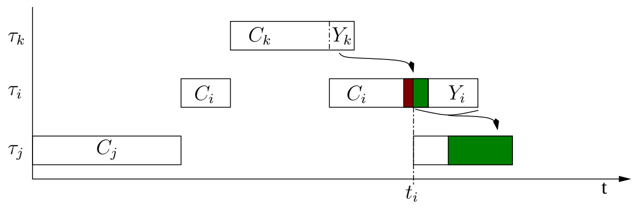

Figure 2. Chained Preemption

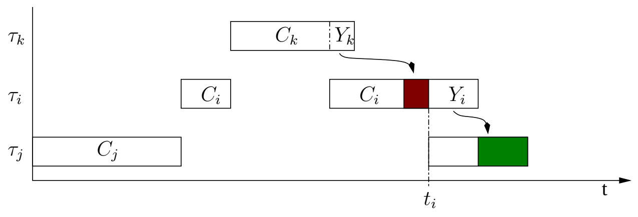

Figure 3. Chained Preemption with Overrun

4.2. Indirect Preemption

An indirect preemption scenario is presented in Figure 2. In this case task τ_k will preempt τ_i. The procedure is similar to the one previously described. Task τ_k budget is increased by Y_k. When the job of τ_i ends (at t_i), Z_i will hold the execution time with the added preemption delay incurred. The value $Y_k - (Z_i - C_i)$ estimates the unused CRPD budget. This leads to task τ_j receiving $Y_k - (Z_i - C_i) + Y_i$. Again the unused budget will be treated as slack in the usual way if there is no task preempted by τ_j.

This simple case is theoretically as sound as the previous one, for the same reason, provided that budgets are not overrun.

4.3. Preemption and Overrun

An interesting case arises when τ_i consumes the entire budget donated by τ_k. This could happen when the resource allocateor assigns smaller budget than required for the completion of the job in a worst-case scenario (depicted in Figure 3). In this situation, only Y_i would be donated to task τ_j. This is potentially incorrect, since some cache misses of task τ_j could have been induced by task τ_k, but τ_j is not getting any compensation for the preemption. This would jeopardise the timing isolation.

The approach should attempt to achieve temporal isolation in the presence of misbehaving applications. There is also a clear hierarchy in terms of timing requirements of hard real-time, soft real-time, and best effort tasks. It has to be noted that it is impossible to distinguish an application which rightfully uses the CRPD donation from a task overusing its budget as the budget provided by the RA has been insufficient to execute this job. We also assume that hard real-time applications are provided with sufficient budget. A useful property of EDF is that preemption relations can be established on the respective deadlines, as a task with a longer relative deadline may not preempt one with a shorter relative deadline [8].

There are fundamental ways to address ambiguous situation decribed in the previous paragraph. Firstly, one could pass instead of task individual Y a global $\bar{Y}$, which covers the worst-case preemption delay. The solution is simple and easy to implement, but in the presence of small tasks and ISRs this would be highly pessimistic. Secondly one could make it mandatory to pass $\bar{Y}$ to a hard real-time task when it is preempted by a non-hard real-time task, and reduced Y for other cases. A disadvantage is that, in this case, the preempting task is either disadvantaged or needs the budget to account for $\bar{Y}$. The latter would have little to no advantage over the first solution. Since the critical hard real-time tasks are expected to have short deadlines, we believe that this we believe that that the approach of indemnifying the hard real-time tasks with $\bar{Y}$ might be a reasonable tradeoff, even if it occasionally violates the temporal isolation properties for non-hard real-time tasks.

Thirdly, when a hard real-time task τ_k is preempted, all further preemption penalties made available while the task τ_k is in the ready queue could be passed to τ_k instead of the preempted task up to a limit of $\bar{Y}$. In some sense this is expected to have similar advantages when compared to the second solution, but the temporal isolation property is now broken for potentially several tasks. On the other hand it is also expected that the number of tasks preempting a hard real-time task is small. Which of the two proposed solutions is suprior is largely dependent on the properties of the task system to be scheduled.

A further problem is illustrated in Figure 4. When a task has overrun, it is reinserted into the ready-queue with the deadline of the new budget. The figure illustrates the evolution of a ready-queue with several significant events. In stage 1, task τ_1 is released and inserted into the ready-queue visible in stage 2 and eventually τ_1 exhausts its current budget. It is then reinserted into the ready-queue in stage 3. On completion of task τ_2 and τ_3, it would obtain its own preemption delay instead of it being passed by τ_3 to τ_4. This violates one of the fundamental principles that a misbehaving task should not impact on other tasks.

To avoid this problem we have devised an alternative relationship for the budget passing: the preemption queue. Whenever a job is first executed, it is placed at the head of the preemption queue. Tasks which complete or run out of budget are removed from the preemption queue. This is indicated in stage 3 and 6

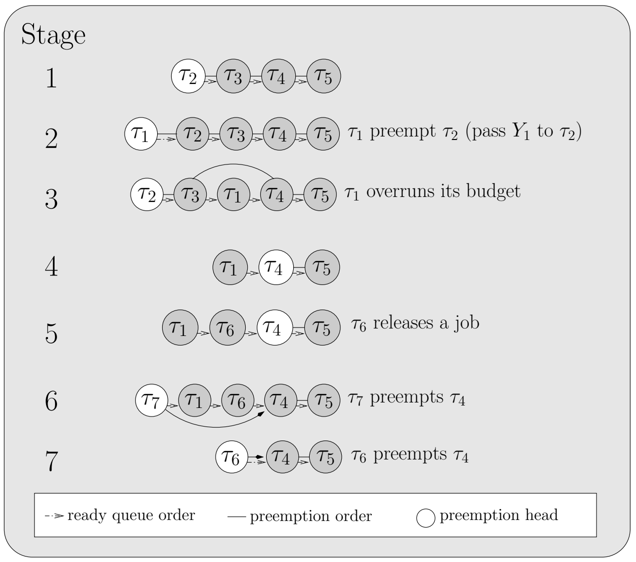
Stage
1
2
3
4
5
6
7
τ_1 preempt τ_2 (pass Y_1 to τ_2)
τ_1 overruns its budget
τ_6 releases a job
τ_7 preempts τ_4
τ_6 preempts τ_4
ready queue order
preemption order
preemption head

Figure 4. Evolution of the Preemption and Ready Queues

of Figure 4. This has two advantages. Firstly, the overhead in maintaining the preemption queue is minimal. Secondly, it avoids assigning preemption delay to jobs which have not executed and therefore have no need for preemption delay compensation. This is depicted for task τ_6 in stages 5 to 7 of Figure 4.

5. Generalisations

Although we have used RBED as reference point for the discussion in the paper, the mechanism may be adopted in other solutions where budgets are assigned for execution such as CBS. Also, while the model used as reference throughout the paper was strictly periodic, there are no assumptions in the solution which require this strict periodicity. As with the related work, the solution provided suits direct-mapped caches naturally. When n-way set-associative caches are present, Y has to be n times the number of sets touched. While we also only used a single cache layer as basis for our discussion, the approach is flexible enough to deal with different sources of preemption delay, like multiple cache levels or branch prediction.

6. Conclusion and Future Work

The presented approach forms the ground work for further work in adaptive preemption delay estimation. This is planned to be performed utilising hardware counters available in many modern processors. In particular we plan to perform measurements to identify the drawbacks and benefits of the different solutions presented in Section 4.3. We expect specific challenges when trying to obtain estimates for the preemption delay in an active system.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *19th RTSS*, pp. 4–13, Dec 1998.

[3] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes," in *24th RTSS*, p. 396, 2003.

[4] Z. Deng and J.-S. Liu, "Scheduling real-time applications in an open environment," in *18th RTSS*, pp. 308–319, Dec 1997.

[5] S. Petters, M. Lawitzky, R. Heffernan, and K. Elphinstone, "Towards real multi-criticality scheduling," in *15th RTCSA*, pp. 155–164, Aug. 2009.

[6] C.-G. Lee, J. 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," *IEEE Transactions on Computers*, vol. 47, pp. 700–713, 1998.

[7] C.-G. Lee, K. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Bounding cache-related preemption delay for real-time systems," *Software Engineering, IEEE Transactions on*, vol. 27, pp. 805–826, Sep 2001.

[8] L. Ju, S. Chakraborty, and A. Roychoudhury, "Accounting for cache-related preemption delay in dynamic priority schedulability analysis," in *DATE 2007*, pp. 1–6, April 2007.

[9] H. Ramaprasad and F. Mueller, "Bounding preemption delay within data cache reference patterns for real-time tasks," in *12th RTAS*, pp. 71–80, April 2006.

[10] H. Ramaprasad and F. Mueller, "Tightening the bounds on feasible preemption points," in *27th RTSS*, pp. 212–224, 2006.

[11] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo, "Cache-aware scheduling with limited preemptions," tech. rep., SSSUP, Pisa, Italy. http://feanor.sssup.it/∼marko/LP RTSS09.pdf accessed on 10th of February, 2010.

[12] S. Altmeyer and G. Gebhard, "Wcet analysis for preemptive scheduling," in *8th WCET*, 2008. Austrian Computer Society (OCG), ISBN 978-3-85403-237-3.

[13] S. Altmeyer and C. Burguiere, "A new notion of useful cache block to improve the bounds of cache-related preemption delay," in *21th ECRTS*, 2009.

[14] C. Lin and S. Brandt, "Improving soft real-time performance through better slack reclaiming," in *26th RTSS*, pp. 12 pp.–421, Dec. 2005.