Accounting for cache related pre-emption delays in hierarchical scheduling with local EDF scheduler

Will Lunniss, Sebastian Altmeyer, Robert I. Davis

Angaben zur Veröffentlichung / Publication details:

Lunniss, Will, Sebastian Altmeyer, and Robert I. Davis. 2014. "Accounting for cache related pre-emption delays in hierarchical scheduling with local EDF scheduler." In Proceedings of the 8th Junior Researcher Workshop on Real-Time Computing - JRWRTC 2014, Versailles, France, October 8-10, 2014, edited by Dorin Maxim and Geoffrey Nelissen, 13–16. CISTER.

Accounting for Cache Related Pre-emption Delays in Hierarchical Scheduling with Local EDF Scheduler

Will Lunniss
Department of Computer Science
University of York
York, UK
wl510@york.ac.uk

Sebastian Altmeyer
Computer Systems Architecture Group
University of Amsterdam
Netherlands
altmeyer@uva.nl

Robert I. Davis
Department of Computer Science
University of York
York, UK
rob.davis@york.ac.uk

ABSTRACT

Hierarchical scheduling provides a means of composing multiple real-time applications onto a single processor such that the temporal requirements of each application are met. This has become a popular technique in industry as it allows applications from multiple vendors as well as legacy applications to co-exist in isolation on the same platform. However, performance enhancing features such as caches mean that one application can interfere with another by evicting blocks from cache that were in use by another application, violating the requirement of temporal isolation. In this paper, we present some initial analysis that bounds the additional delay due to blocks being evicted from cache by other applications in a system using hierarchical scheduling when using a local EDF scheduler.

1. INTRODUCTION

Hierarchical scheduling provides a means of composing multiple applications onto a single processor such that the temporal requirements of each application are met. This is driven by the need to re-use legacy applications that once ran on slower, but dedicated processors. Each application, referred to as a component, has a dedicated server. A global scheduler then allocates processor time to each server, during which the associated component can use its own local scheduler to schedule its tasks.

In hard real-time systems, the schedulability of each task must be known offline in order to verify that the timing requirements will be met at runtime. However, in pre-emptive multi-tasking systems, caches introduce additional *cache related pre-emption delays* (CRPD) caused by the need to re-fetch cache blocks belonging to the pre-empted task which were evicted from the cache by the pre-empting task. These CRPD effectively increase the worst-case execution time of the tasks. It is therefore important to be able to calculate, and account for, CRPD when determining if a system is schedulable or not. This is further complicated when using hierarchical scheduling as servers will often be suspended while their components' tasks are still active, that is they have started but have not yet completed execution. While a server is suspended, the cache can be polluted by the tasks belonging to other components. When the global scheduler then switches back to the first server, tasks belonging to the associated component may have to reload blocks into cache that were in use before the global context switch.

Hierarchical scheduling has been studied extensively in the past 15 years. Deng and Liu [7] were the first to propose such a two-level scheduling approach. Later Feng and Mok [8] proposed the resource partition model and schedulability analysis based on the supply bound function. Shin and Lee [16] introduced the concept of a temporal interface and the periodic resource model, and refined the analysis of Feng and Mok. When using a local EDF scheduler, Lipari *et al.* [11] [12] have investigated allocating server capacity to components, proposing an exact solution. Recently Fisher and Dewan [9] have developed a polynomial-time approximation with minimal over provisioning of resources.

Hierarchical systems have been used mainly in the avionics industry. For example, the ARINC 653 standard [2] defines temporal partitioning for avionics applications. The global scheduler is a simple Time Division Multiplexing (TDM), in which time is divided into frames of fixed length, each frame is divided into slots and each slot is assigned to one application.

Analysis of CRPD uses the concept of *useful cache blocks* (UCBs) and *evicting cache blocks* (ECBs) based on the work by Lee *et al.* [10]. Any memory block that is accessed by a task while executing is classified as an ECB, as accessing that block may evict a cache block of a pre-empted task. Out of the set of ECBs, some of them may also be UCBs. A memory block m is classified as a UCB at program point ρ, if (i) m may be cached at ρ and (ii) m may be reused at program point q that may be reached from ρ without eviction of m on this path. In the case of a pre-emption at program point ρ, only the memory blocks that are (i) in cache and (ii) will be reused, may cause additional reloads. For a more thorough explanation of UCBs and ECBs, see section 2.1 "Pre-emption costs" of [1].

A number of approaches have been developed for calculating the CRPD when using fixed priority pre-emptive scheduling under a flat, single-level system. A summary of these approaches, along with the state-of-the-art approach is available in [1]. In 2013, Lunniss *et al.* [14] presented a number of approaches for calculating CRPD when using pre-emptive EDF scheduling.

In 2014, Lunniss *et al.* [13] extended previous works to include CRPD analysis under hierarchical scheduling when using a local FP scheduler.

The remainder of the paper is organised as follows. Section 2 introduces the system model, terminology and notation used. Section 3 recaps existing CRPD and schedulability analysis. Section 4 introduces the new analysis for calculating component level CRPD incurred in hierarchical systems when using a local EDF scheduler. In section 5 the analysis is evaluated, and section 6 concludes with a summary and outline of future work.

2. SYSTEM MODEL

We assume a single processor system comprising m components, each with a dedicated server ($S^1..S^m$) that allocates processor capacity to it. We use Ψ to represent the set of all components in the system. G is used to indicate the index of the component that is being analysed. Each server S^G has a budget Q^G and a period P^G, such that the associated component will receive Q^G units of execution time from its server every P^G units of time. Servers are assumed to be scheduled globally using a non-pre-emptive scheduler, as found in systems that use time partitioning to divide up access to the processor. While a server has remaining capacity and is allocated the processor, we assume that the tasks of the associated component are scheduled using pre-emptive EDF.

The system comprises a taskset Γ made up of a fixed number of tasks ($\tau_1..\tau_n$) divided between the components. Each component contains a strict subset of the tasks, represented by Γ^G. For simplicity, we assume that the tasks are independent and do not share resources requiring mutually exclusive access, other than the processor.

Each task, τ_i may produce a potentially infinite stream of jobs that are separated by a minimum inter-arrival time or period T_i. Each task has a relative deadline D_i, a worst case execution time C_i (determined for non-pre-emptive execution). We assume that deadlines are either *implicit* (i.e. $D_i=T_i$) or *constrained* (i.e. $D_i \leq T_i$).

Each task τ_i has a set of UCBs, UCB_i and a set of ECBs, ECB_i represented by a set of integers. If for example, task τ_1 contains 4 ECBs, where the second and fourth ECBs are also UCBs, these can be represented using $ECB_1 = \{1,2,3,4\}$ and $UCB_1 = \{2,4\}$. Each component *G* also has a set of UCBs, UCB^G and a set of ECBs, ECB^G, that contain respectively all of the UCBs, and all of the ECBs, of their tasks, i.e. $UCB^G = \bigcup_{\forall \tau i \in \Gamma^G} UCB_i$ and $ECB^G = \bigcup_{\forall \tau i \in \Gamma^G} ECB_i$.

Each time a cache block is reloaded, a cost is introduced that is equal to the *block reload time* (BRT). We assume a direct mapped cache, but the work extends to set-associative caches with the LRU replacement policy as described in section 2 of [1]. We focus on instruction only caches.

3. EXISTING SCHEDULABILITY AND CRPD ANALYSIS

Schedulability analysis for EDF uses the *processor demand bound* function [3], [4], in order to determine the demand on the processor within a fixed interval. It calculates the maximum execution time requirement of all tasks' jobs which have both their arrival times and their deadlines in a contiguous interval of length *t*. Baruah *et al.* showed that a taskset is schedulable under EDF iff $\forall t > 0, h(t) \leq t$. We use a modified equation for $h(t)$ from [14] which includes $\gamma_{t,j}$ to represent the CRPD caused by task τ_j that may affect any job of a task with both its release times and absolute deadlines within an interval of length *t*.

$$h(t) = \sum_{j=1}^{n} \left(\max\left\{0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right\} C_j + \gamma_{t,j} \right) \quad (1)$$

In order to determine the schedulability of a taskset in a hierarchical system, we must account for the limited access to the processor. The *supply bound function* [16], or specifically the inverse of it, can be used to determine the maximum amount of time needed by a specific server to supply some capacity *c*. We define the *inverse supply bound function*, *isbf*, for component *G* as $isbf^G$ [15]:

$$isbf^G(c) = c + (P^G - Q^G)\left(\left\lceil \frac{c}{Q^G} \right\rceil + 1 \right) \quad (2)$$

4. NEW CRPD ANALYSIS

In [13] Lunniss *et al.* presented a number of approaches for calculating CRPD in hierarchical systems when using a local FP scheduler. We now describe how CRPD analysis can be adapted for use with a local EDF scheduler. This analysis assumes a non-pre-emptive global scheduler (i.e. the capacity of a server is supplied without pre-emption, but may be supplied starting at any time during the server's period).

The analysis must account for the cost of reloading any UCBs into cache that may be evicted by tasks running in the other components, which we call component level CRPD. To account for the component level CRPD, we define a new term γ_t^G that represents the CRPD incurred by tasks in component *G* due to tasks in the other components running while the server (S^G) for component *G* is suspended. Combining (1), with $isbf^G$, (2), and γ_t^G, we get the following expression for determining the processor demand within an interval of length *t*.

$$h(t) = isbf^G\left(\sum_{j=1}^{n} \left(\max\left\{0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right\} C_j + \gamma_{t,j} \right) + \gamma_t^G \right) \quad (3)$$

In the computation of γ_t^G, we use a number of terms, described below. We use $E_j(t)$ to denote the maximum number of jobs of task τ_j that can have both their release times and their deadlines in an interval of length *t*, which we calculate as follows:

$$E_j(t) = \max\left(0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor\right) \quad (4)$$

We use $E^G(t)$ to denote the maximum number of times server S^G can be both suspended and resumed during *t*. Note that (5) can be used with $t=D_j$ to calculate the maximum number of times server S^G can be suspended and resumed during a single job of task τ_j.

$$E^G(t) = 1 + \left\lfloor \frac{t}{P^G} \right\rfloor \quad (5)$$

We use the term *disruptive execution* to describe an execution of server S^Z while server S^G is suspended that results in tasks from component Z evicting cache blocks that tasks in component G might have loaded and may need to reload in an interval of length *t*. Note that if server S^Z runs more than once while server S^G is suspended, its tasks cannot evict the same blocks twice and as such, the number of disruptive executions is bounded by the number of times that server S^G can be both suspended and resumed. Specifically, we are interested in how many disruptive executions a server can have during an interval of length *t*. We use X^Z to denote the maximum number of such disruptive executions.

$$X^Z(S^G, t) = \min\left(E^G(t), 1 + \left\lceil \frac{t}{P^Z} \right\rceil\right) \quad (6)$$

4.1 Component level CRPD

We first calculate an upper bound on the UCBs that if evicted by tasks in the other components may need to be reloaded. We do this by forming a multiset that contains the UCBs of task τ_k repeated $E^G(D_k)E_k(t)$ times for each task in $\tau_k \in \Gamma^G$. This multiset reflects the fact that server S^G can be suspended and resumed at most $E^G(D_k)$ times during a single job of task τ_k and there can be at most $E_k(t)$ jobs of task τ_k that have their release times and absolute deadlines within the interval of length *t*.

$$M_{G,t}^{ucb} = \bigcup_{\forall k \in \Gamma^G} \left(\bigcup_{E^G(D_k)E_k(t)} UCB_k \right) \quad (7)$$

The second step is to determine which ECBs of the tasks in the other components could evict the UCBs in (7), for which we present three different approaches.

4.1.1 UCB-ECB-Multiset-All

The first option is to assume that every time server S^G is suspended, all of the other servers run and their tasks evict all the cache blocks that they use. We therefore take the union of all ECBs belonging to the other components to get the set of blocks that could be evicted. We form a second multiset that contains $E^G(t)$ copies of the ECBs of all of the other components in the system. This multiset reflects the fact that the other servers' tasks can evict blocks (that need to be reloaded) at most $E^G(t)$ times within an interval of length *t*.

$$M_{G,t}^{ecb-A} = \bigcup_{E^G(t)} \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \text{ECB}^Z \right) \quad (8)$$

The total CRPD incurred by tasks in component G due to the other components in the system is then given by the size of the multiset intersection of $M_{G,t}^{ucb}$ (7) and $M_{G,t}^{ecb-A}$ (8).

$$\gamma_t^G = \text{BRT} \bullet \left| M_{G,t}^{ucb} \cap M_{G,t}^{ecb-A} \right| \quad (9)$$

4.1.2 UCB-ECB-Multiset-Counted

The above approach works well when the global scheduler uses a TDM schedule such that each server has the same period and/or components share a large number of ECBs. If some servers run less frequently than server S^G, then the number of times that their ECBs can evict blocks may be over counted. One solution to this problem is to consider each component separately by calculating the number of disruptive executions, $X^Z\left(S^G,t\right)$, that server S^Z can have on tasks in component G during t. We form a second multiset that contains $X^Z\left(S^G,t\right)$ copies of ECB^Z for each of the other components Z in the system. This multiset reflects the fact that the tasks of each component Z can evict blocks at most $X^Z\left(S^G,t\right)$ times within an interval of length t.

$$M_{G,t}^{ecb-C} = \bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(\bigcup_{X^Z\left(S^G,t\right)} \text{ECB}^Z \right) \quad (10)$$

The total CRPD incurred by task τ_i, in component G due to the other components in the system is then given by the size of the multiset intersection of $M_{G,t}^{ucb}$ (7) and $M_{G,t}^{ecb-C}$ (10).

$$\gamma_t^G = \text{BRT} \bullet \left| M_{G,t}^{ucb} \cap M_{G,t}^{ecb-C} \right| \quad (11)$$

4.1.3 UCB-ECB-Multiset-Open

In *open* hierarchical systems, the other components may not be known a priori as they can be introduced into a system dynamically. Additionally, even in *closed* systems, full information about the other components in the system may not be available until the final stages of system integration. However, as the cache utilisation of the other components can often be greater than the size of the cache, the precise set of ECBs does not matter. We form a second multiset that contains $E^G(t)$ copies of all cache blocks. This multiset reflects the fact that server S^G can be both suspended and then resumed, after the entire contents of the cache have been evicted at most $E^G(t)$ times within an interval of length t.

$$M_{G,t}^{ecb-O} = \bigcup_{E^G(t)} \left(\{1,2,..N\} \right) \quad (12)$$

Where N is the number of cache sets.

The total CRPD incurred by tasks in component G due to the other unknown components in the system is then given by the size of the multiset intersection of $M_{G,t}^{ucb}$ (7) and $M_{G,t}^{ecb-O}$ (12).

$$\gamma_t^G = \text{BRT} \bullet \left| M_{G,t}^{ucb} \cap M_{G,t}^{ecb-O} \right| \quad (13)$$

For all approaches, we calculated the limit (largest value of t that needs to be checked in (1)) using an inflated utilisation in a similar way to that described in section *V. D* of [14].

5. EVALUATION

In this section we compare the different approaches for calculating CRPD in hierarchical scheduling using synthetically generated tasksets. The evaluation was setup to model an ARM processor clocked at 100MHz with a 2KB direct-mapped instruction cache. The cache was setup with a line size of 8 Bytes, giving 256 cache sets, 4 Byte instructions, and a BRT of 8µs. To generate the components and tasksets, we generated n (default of 24) tasks using the UUnifast algorithm [6] to calculate the utilisation, U_i of each task so that the utilisations added up to the desired utilisation level. Periods T_i, were generated at random between 10ms and 1000ms according to a log-uniform distribution. C_i was then calculated via $C_i = U_i T_i$. We assigned implicit deadlines, i.e. $D_i = T_i$. We used the UUnifast algorithm to obtain the number of ECBs for each task so that the ECBs added up to the desired cache utilisation (default of 10). Here, cache utilisation describes the ratio of the total size of the tasks to the size of the cache. A cache utilisation of 1 means that the tasks fit exactly in the cache, whereas a cache utilisation of 10 means the total size of the tasks is 10 times the size of the cache. The number of UCBs was chosen at random between 0 and 30% of the number of ECBs on a per task basis, and the UCBs were placed in a single group at a random location in each task. We then split the tasks at random into 3 components with equal numbers of tasks in each and set the period of each component's server to 5ms. We generated 1000 systems using this technique.

For each system, the total task utilization across all tasks not including pre-emption cost was varied from 0.025 to 1 in steps of 0.025. For each utilization value, we initialised each servers' capacity to the minimum possible value, (i.e. the utilisation of all of its tasks). We then performed a binary search between this minimum and the maximum, (i.e. 1 minus the minimum utilisation of all of the other components) until we found the server capacity required to make the component schedulable. As the servers all had equal periods, provided all components were schedulable and the total capacity required by all servers was ≤ 100%, then the system was deemed schedulable at that specific utilisation level. For every approach, the intra-component CRPD (between tasks in the same component) was calculated using the Combined Multiset approach given by Lunniss *et al.* [14].

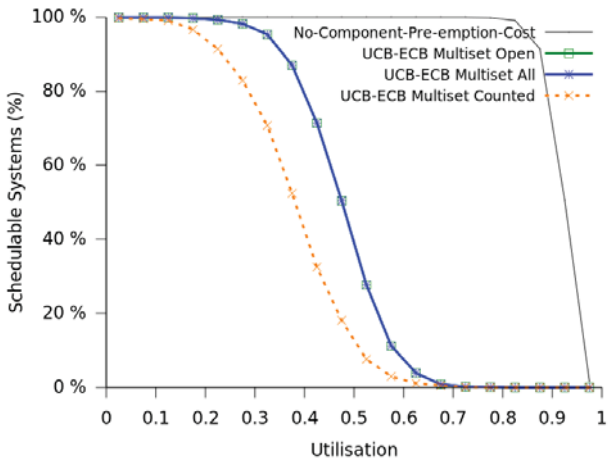

Figure 1. Percentage of systems deemed schedulable

Figure 1 shows that the UCB-ECB-Multiset-All and UCB-ECB-Multiset-Open approaches deem the same number of tasksets schedulable. This is due to the cache utilisation of the other components being greater than the size of the cache, which causes the set of ECBs to be equal, i.e. contain all cache blocks. The UCB-ECB-Multiset-Counted approach deems a lower number of tasksets schedulable because it considers the effects of the other components individually. As the components have equal server periods, each time a component is suspended, it is assumed that each other component will evict it's set of ECBs, when in fact

they may only be evicted once per suspension. We note that the results show that the analysis is somewhat pessimistic, as there is a large difference between the No-Component-Pre-emption-Cost case, and the approaches that consider component pre-emption costs. Examining equation (7), we note that $E^G(D_k)E_k(t)$ is based on the deadline of a task and as such, the analysis effectively assumes the UCBs of all tasks in component G could be in use each time the server for component G is suspended.

The server period is a critical parameter when composing a hierarchical system. The results for varying the server period from 1ms to 20ms, with a fixed range of task periods from 10 to 1000ms are shown in Figure 2 using the weighted schedulability measure [5]. When the component pre-emption costs are ignored, having a small server period ensures that short deadline tasks meet their time constraints. However, switching between components clearly has a cost associated with it making it desirable to switch as infrequently as possible. As the server period increases, schedulability increases due to a smaller number of server context switches, and hence component CRPD, up until around 7-8ms for the best performance. At this point, although the component CRPD continues to decrease, short deadline tasks start to miss their deadlines due to the delay in server capacity being supplied unless server capacities are greatly inflated, and hence the overall schedulability of the system decreases.

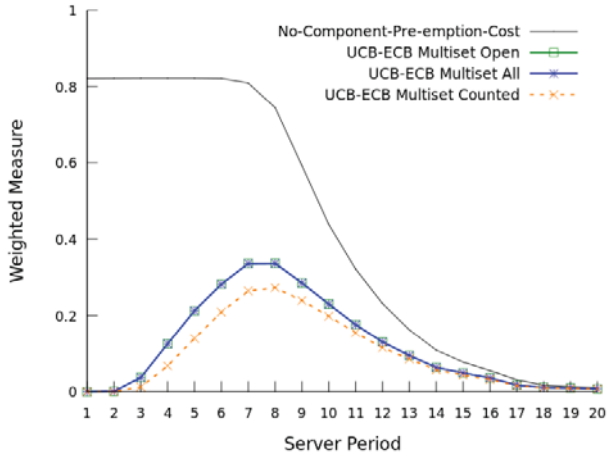

Figure 2. Weighted measure of the schedulability when varying the server period from 1 to 20ms

6. CONCLUSION

In this paper, we have presented some initial analysis for bounding CRPD under hierarchical scheduling when using a local EDF scheduler. This analysis builds on existing work for determining CRPD under single-level EDF scheduling [14], and hierarchical scheduling with a local FP scheduler [13]. We also showed that when taking inter-component CRPD into account, minimising server periods does not maximise schedulability. Instead, the server period must be carefully selected to minimise inter-component CRPD while still ensuring short deadline tasks meet their time constraints. We note that the analysis is somewhat pessimistic due to the use of a tasks' deadline for determining how many times its component could be suspended and resumed during its execution. In future work we would like to investigate ways to resolve this. Furthermore, we believe that the analysis could be optimised when using harmonic server periods, which could lead to an improvement in the UCB-ECB-Multiset-Counted approach. Finally, we would like to extend the analysis for use with a pre-emptive global scheduler.

ACKNOWLEDGEMENTS

This work was partially funded by the UK EPSRC through the Engineering Doctorate Centre in Large-Scale Complex IT Systems (EP/F501374/1), the UK EPSRC funded MCC (EP/K011626/1), the European Community's ARTEMIS Programme and UK Technology Strategy Board, under ARTEMIS grant agreement 295371-2 CRAFTERS, and COST Action IC1202: Timing Analysis On Code-Level (TACLe).

REFERNECES

[1] Altmeyer, S., Davis, R.I., and Maiza, C. Improved Cache Related Pre-emption Delay Aware Response Time Analysis for Fixed Priority Pre-emptive Systems. *Real-Time Systems*, 48, 5 (2012), 499-512.

[2] ARINC. *ARINC 653: Avionics Application Software Standard Interface (Draft 15)*. Airlines Electronic Engineering Committee (AEEC), 1996.

[3] Baruah, S. K., Mok, A. K., and Rosier, L. E. Preemptive Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS)* (1990), 182-190.

[4] Baruah, S. K., Rosier, L. E., and Howell, R. R. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor. *Real-Time Systems*, 2, 4 (1990), 301-324.

[5] Bastoni, A., Brandenburg, B., and Anderson, J. Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability. In *Proceedings of Operating Systems Platforms for Embedded Real-Time applications (OSPERT)* (2010), 33-44.

[6] Bini, E. and Buttazzo, G. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, 30, 1 (2005), 129-154.

[7] Deng, Z. and Liu, J. W. S. Scheduling Real-Time Applications in Open Environment. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)* (1997).

[8] Feng, X. and Mok, A. K. A Model of Hierarchical Real-Time Virtual Resources. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)* (2002), 26-35.

[9] Fisher, N. and Dewan, F. A Bandwidth Allocation Scheme for Compositional Real-time Systems with Periodic Resources. *Real-Time Systems*, 48, 3 (2012), 223-263.

[10] Lee, C., Hahn, J., Seo, Y. et al. Analysis of Cache-related Preemption Delay in Fixed-priority Preemptive Scheduling. *IEEE Transactions on Computers*, 47, 6 (June 1998), 700-713.

[11] Lipari, G. and Baruah, S. K. Efficient Scheduling of Real-time Multi-task Applications in Dynamic Systems. In *Proceddings Real-Time Technology and Applications Symposium (RTAS)* (2000), 166-175.

[12] Lipari, G., Carpenter, J., and Baruah, S. A Framework for Achieving Inter-application Isolation in Multiprogrammed, Hard Real-time Environments. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS)* (2000), 217-226.

[13] Lunniss, W., Altmeyer, S., Lipari, G., and Davis, R. I. Accounting for Cache Related pre-emption Delays in Hierarchical Scheduling. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems (RTNS)* (2014).

[14] Lunniss, W., Altmeyer, S., Maiza, C., and Davis, R. I. Intergrating Cache Related Pre-emption Delay Analysis into EDF Scheduling. In *Proceedings 19th IEEE Converence on Real-Time and Embedded Technology and Applications (RTAS)* (2013), 75-84.

[15] Richter, K. *Compositional Scheduling Analysis Using Standard Event Models*. PhD Dissertation, Technical University Carolo-Wilhelmina of Braunschweig, 2005.

[16] Shin, I. and Lee, I. Periodic Resource Model for Compositional Real-Time Guarantees. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS)* (2003), 2-13.