Accounting for Cache Related Pre-emption Delays in Hierarchical Scheduling with Local EDF Scheduler
Lunniss, W.; Altmeyer, S.J.; Davis, R.I.

Published in:

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
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 η 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 (S1,…,Sm) 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 Si has a budget Qi and a period Pi, such that the associated component will receive Qi units of execution time from its server every Pi 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.

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.
The system comprises a taskset Τ made up of a fixed number of tasks (τ₁,τ₂) divided between the components. Each component contains a strict subset of the tasks, represented by τG. For simplicity, we assume that the tasks are independent and do not share resources requiring mutually exclusive access, other than the processor.

Each task, τi, 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 Di, a worst case execution time Di (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≤T_i).

Each task τi has a set of UCBs, UCB_i, and a set of ECBs, ECB_i, represented by a set of integers. If for example, task τi contains 4 ECBs, where the second and fourth ECBs are also UCBs, these can be represented using ECB_i = {1,2,3,4} and UCB_i = {2,4}. Each component G also has a set of UCBs, UCBG and a set of ECBs, ECBG, that contain respectively all of the UCBs, and all of the ECBs, of its tasks, i.e. UCBG = \bigcup_{i \in \text{tasks of } G} \text{UCB}_i and ECBG = \bigcup_{i \in \text{tasks of } G} \text{ECB}_i.

Each time a cache block is reloaded, a cost is introduced that is the sum of the worst case execution time of all the tasks in the cache block. 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 ∀τi > 0, h(t) ≤ t. We use the modified equation for h(t) from [14] which includes γi,t to represent the CRPD caused by task τi that may affect any job of a task with both its release times and absolute deadlines within an interval of length t.

\[
h(t) = \sum_{i=1}^{n} \max\left(0,1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i + \gamma_{i,t} \tag{1}
\]

In order to determine the schedulability of a taskset in a hierarchical system, we must account for the limited access to the processor.

4. NEW CRPD ANALYSIS

In [3] 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 γG,t that represents the CRPD incurred by tasks in component G due to tasks in the other components running while the server (SG) for component G is suspended. Combining (1), with ishfG, (2), and γG,t, we get the following expression for determining the processor demand within an interval of length t.

\[
h(t) = ishfG \left( \sum_{i=1}^{n} \max\left(0,1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i + \gamma_{i,t} \right) + \gamma_G \tag{3}
\]

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

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

We use E^G_j(t) to denote the maximum number of times server SG 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 SG can be suspended and resumed during a single job of task τj.

\[
E^G_j(t) = \max\left(0,1 + \left\lfloor \frac{t}{p^2} \right\rfloor \right) \tag{5}
\]

We use the term disruptive execution to describe an execution of server SG 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^G runs more than once while server SG 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 SG 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^G to denote the maximum number of such disruptive executions.

\[
X^G \leq \min\left(E^G_j(t), \left\lfloor \frac{t}{p^2} \right\rfloor \right) \tag{6}
\]

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 τi, repeated E^G_i(D_i,P^G) times for each task in τi ∈ Γ^G. This multiset reflects the fact that server SG can be suspended and resumed at most E^G_i(D_i,P^G) times during a single job of task τi and there can be at most E_j(t) jobs of task τi that have their release times and absolute deadlines within the interval of length t.

\[
M_{\text{UCB}}^p = \bigcup_{\forall \tau \in \text{tasks in } G} \left( \bigcup_{\forall \tau_i \in \text{tasks in } G} \text{UCB}_i \right) \tag{7}
\]

The second step is to determine which UCBS 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 SG 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 UCBS belonging to the other components to get the set of blocks that could be evicted. We form a second multiset that contains E^G_j(t) copies of the UCBS 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_j(t) times within an interval of length t.
\[ M^{ecb\cdot A}_{G_j} = \bigcup_{E^G(t)} \bigcup_{z \in Z} \{ ECB^B \} \]

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 \( M^{dbc}_{G_j} \) and \( M^{ecb\cdot A}_{G_j} \).

\[ \gamma_G^I = \text{BRT} \cdot M^{ucb \cap M^{ecb\cdot A}}_{G_j} \]

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, \( X^G[S^G, t] \), that server \( S^G \) can have on tasks in component \( G \) during \( t \). We form a second multiset that contains \( X^G[S^G, t] \) copies of ECB\(^B\) 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 \( X^Z[S^G, t] \) times within an interval of length \( t \).

\[ M^{ecb\cdot C}_{G_j} = \bigcup_{z \in Z} \bigcup_{x} \{ ECB^B \} \]

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 \( M^{dbc}_{G_j} \) and \( M^{ecb\cdot C}_{G_j} \).

\[ \gamma_G^I = \text{BRT} \cdot M^{ucb \cap M^{ecb\cdot C}}_{G_j} \]

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 the 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 \( t \).

\[ M^{ecb\cdot O}_{G_j} = \bigcup_{E^G(t)} \{ \{1, 2, \ldots, N \} \} \]

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 \( M^{dbc}_{G_j} \) and \( M^{ecb\cdot O}_{G_j} \).

\[ \gamma_G^I = \text{BRT} \cdot M^{ucb \cap M^{ecb\cdot O}}_{G_j} \]

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 8\( \mu s \). 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 \( C_i = U_i \cdot T_i \). 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 utilisation 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 \( \leq 100\% \), 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 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 $E_{D_G}(D, E, v)$ 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 10000ms 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).

### REFERENCES


