Cache related pre-emption delays in hierarchical scheduling

Will Lunniss[1] · **Sebastian Altmeyer**[2] · **Giuseppe Lipari**[3,4] · **Robert I. Davis**[1]

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 analysis that bounds the additional delay due to blocks being evicted from cache by other applications in a system using hierarchical scheduling when using either a local FP or EDF scheduler.

Keywords Cache related pre-emption delays · Hierarchical scheduling · Fixed priority pre-emptive scheduling · Earliest deadline first pre-emptive scheduling · Response time analysis

✉ Will Lunniss
wl510@york.ac.uk

Sebastian Altmeyer
sebastian.altmeyer@uni.lu

Giuseppe Lipari
giuseppe.lipari@univ-lille1.fr

Robert I. Davis
rob.davis@york.ac.uk

[1] Department of Computer Science, University of York, York, UK

[2] LASSY Group, University of Luxembourg, Luxembourg, Luxembourg

[3] CRIStAL, UMR 9189, Univ. Lille, 59650 Villeneuve d'Ascq, France

[4] IRCICA, USR 3380, 59650 Villeneuve d'Ascq, France

1 Introduction

There is a growing need in industry to combine multiple applications together to build complex embedded real-time systems. This is driven by the need to re-use legacy applications that once ran on slower, but dedicated processors. Typically, it is too costly to go back to the design phase resulting in a need to use applications as-is. Furthermore, there are often a number of vendors involved in implementing today's complex embedded real-time systems, each supplying separate applications which must then be integrated together. Hierarchical scheduling provides a means of composing multiple applications onto a single processor, such that the temporal requirements of each application are met. Each application, or 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 *worst-case execution time* (WCET) 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, which violates the assumption used by classical schedulability analysis that a tasks' execution time is not affected by the other tasks in the system. It is therefore important to be able to calculate, and account for, CRPD when determining if a system is schedulable, otherwise the results obtained could be optimistic. This is further complicated when using hierarchical scheduling as servers will often be suspended while their components' tasks are still active. In this case they have started, but have not yet completed executing. 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.

1.1 Related work on hierarchical scheduling

Hierarchical scheduling has been studied extensively in the past 15 years. Deng and Liu (1997) were the first to propose such a two-level scheduling approach. Later Feng and Mok (2002) proposed the resource partition model and schedulability analysis based on the supply bound function. Shin and Lee (2013) introduced the concept of a temporal interface and the periodic resource model, and refined the analysis of Feng and Mok. Kuo and Li (1998) and Saewong et al. (2002) specifically focused on fixed priority hierarchical scheduling. Lipari and Bin (2005) solved the problem of computing the values of the partition parameters to make an application schedulable. Davis and Burns (2005) proposed a method to compute the response time of tasks running on a local fixed priority scheduler. Later, Davis and Burns (2008) investigated selecting optimal server parameters for fixed priority pre-emptive hierarchical systems. When using a local EDF scheduler Lipari et al. (2000a, b) investigated allocating server capacity to components, proposing an exact solution. Recently Fisher and Dewan

(2012) developed a polynomial-time approximation with minimal over provisioning of resources.

Hierarchical systems have been used mainly in the avionics industry. Integrated Modular Avionics (IMA) (Watkins and Walter 2007; ARINC 1991) is a set of standard specifications for simplifying the development of avionics software. Among other requirements it allows different independent applications to share the same hardware and software resources (ARINC 1996). The ARINC 653 standard (ARINC 1996) 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.

1.2 Related work on CRPD

Analysis of CRPD uses the concept of *useful cache blocks* (UCBs) and *evicting cache blocks* (ECBs) based on the work by Lee et al. (1998). 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, assuming no pre-emption. 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 Sect. 2.1 "Pre-emption costs" of Altmeyer et al. (2012).

Depending on the approach used, the CRPD analysis combines the UCBs belonging to the pre-empted task(s) with the ECBs of the pre-empting task(s). Using this information, the total number of blocks that are evicted, which must then be reloaded after the pre-emption, can be calculated and combined with the cost of reloading a block to give an upper bound on the CRPD.

A number of approaches have been developed for calculating the CRPD when using FP pre-emptive scheduling under a single-level system. They include Lee et al. (1998) UCB-Only approach, which considers just the pre-empted task(s), and Busquets-Mataix et al. (1996) ECB-Only approach which considers just the pre-empting task. Approaches that consider the pre-empted and pre-empting task(s) include Tan and Mooney (2007) UCB-Union approach, Altmeyer et al. (2011) ECB-Union approach, and an alternative approach by Staschulat et al. (2005). Finally, there are advanced multiset based approaches that consider the pre-empted and pre-empting task(s) by Altmeyer et al. (2012), ECB-Union Multiset, UCB-Union Multiset, and a combined multiset approach. There has been less work towards developing CRPD analysis for EDF pre-emptive scheduling under a single-level system. Campoy et al. (2004) proposed an approach where the majority of blocks are locked into cache, with a small temporal buffer for use by those that are not. An upper bound on the CRPD can then be calculated by including the relatively small cost of reloading the temporal buffer for each pre-emption that could occur. Ju et al. (2007) considered the intersection of the pre-empted task's UCBs with the pre-empting task's ECBs. Lunniss et al. (2013) adapted a number of approaches for calculating CRPD for FP to work with EDF.

Including the ECB-Only, UCB-Only, UCB-Union, ECB-Union, ECB-Union Multiset, UCB-Union Multiset and combined multiset CRPD analysis for FP given by Busquets et al. (1996), Lee et al. (1998), Tan and Mooney (2007), and Altmeyer et al. (2012).

Xu et al. (2013) proposed an approach for accounting for cache effects in multicore virtualization platforms. However, their focus was on how to include CRPD and *cache related migration delays* into a compositional analysis framework, rather than how to tightly bound the task and component CRPD.

This paper forms an extended version of "Accounting for Cache Related Pre-emption Delays in Hierarchical Scheduling" (Lunniss et al. 2014a) which was published in RTNS 2014. The main additional contributions are as follows: The introduction of new CRPD analysis for hierarchical scheduling with a local EDF scheduler (Sect. 7). Extending the evaluation with results for EDF and with some additional evaluations to explain the differences in the results obtained using a local FP scheduler vs a local EDF scheduler. Note: Preliminary results for the EDF extension were presented in a workshop paper "Accounting for Cache Related Pre-emption Delays in Hierarchical Scheduling with Local EDF Scheduler" (Lunniss et al. 2014b) at the JRWRTC at RTNS 2014.

1.3 Organisation

The remainder of the paper is organised as follows. Section 2 introduces the system model, terminology and notation used. Section 3 covers existing schedulability and CRPD analysis for single-level systems scheduled under FP. Section 4 details how schedulability analysis can be extended for hierarchical systems. Section 5 introduces the new analysis for accounting for CRPD in hierarchical scheduling with a local FP scheduler. Section 6 covers existing schedulability and CRPD analysis for single-level systems scheduled under EDF. Section 7 introduces the new analysis for accounting for CRPD in hierarchical scheduling with a local EDF scheduler. Section 8 evaluates the analysis using case study data, and Sect. 9 evaluates it using synthetically generated tasksets. Finally, Sect. 9.4 concludes with a summary and outline of future work.

2 System model, terminology and notation

This section describes the system model, terminology, and notation used in the rest of the paper.

We assume a uniprocessor system comprising m applications or components, each with a dedicated server ($S^1 \dots S^m$) 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 according to the local scheduling policy. If there are no tasks in the associated component to schedule, we assume that the processor idles until the server exhausts all of its capacity, or a new task in the associated component is released. We initially assume a *closed* system, whereby information about all components in the system is known. Later we relax this assumption and present approaches that can be applied to *open* hierarchical systems, where the other components may not be known *a priori* as they can be introduced into a system dynamically. These approaches can also be used in cases where full information about the other components in the system may not be available until the final stages of system integration.

The system comprises a taskset Γ made up of a fixed number of tasks $(\tau_1 \ldots \tau_n)$ divided between the components which do not share code. Tasks are scheduled locally using either FP or EDF. In the case of a local FP scheduler, the priority of task τ_i, is i, where a priority of 1 is the highest and n is the lowest. Priorities are unique, but are only meaningful within 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. (We note that global and local resource sharing has been extensively studied for hierarchical systems (Davis and Burns 2006; Behnam et al. 2007; Asberg et al. 2013). Resource sharing and its effects on CRPD have also been studied for single level systems by Altmeyer et al. (2011, 2012). However, such effects are beyond the scope of this paper).

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) and release jitter J_i. Each task has a utilisation U_i, where $U_i = C_i / T_i$, and each taskset has a utilisation U which is equal to the sum of its tasks' utilisations. We assume that deadlines are *constrained* (i.e. $D_i \leq T_i$). In the case of a local FP scheduler, we use the notation hp(i) to mean the set of tasks with priorities higher than that of task τ_i and hep(i) to mean the set of tasks with higher or equal priorities. We also use the notation hp(G,i), and hep(G,i), to restrict hp(i), and hep(i), to just tasks of component G.

With respect to a given system model, a schedulability test is said to be *sufficient* if every taskset it deems to be schedulable is in fact schedulable. Similarly, a schedulability test is said to be *necessary* if every taskset it deems to be unschedulable is in fact unschedulable. Tests that are both sufficient and necessary are referred to as *exact*.

A schedulability test A is said to *dominate* another schedulability test B if all of the tasksets deemed schedulable by test B are also deemed schedulable by test A, and there exist tasksets that are schedulable according to test A but not according to test B. Schedulability tests A and B are said to be *incomparable* if there exists tasksets that are deemed schedulable by test A and not by test B and also tasksets that are deemed schedulable by test B and not by test A.

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 $\mathrm{ECB}_1 = \{1,2,3,4\}$ and $\mathrm{UCB}_1 = \{2,4\}$. We use |UCB| to give the cardinality of the set, for example if $\mathrm{UCB}_1 = \{2,4\}$, $|\mathrm{UCB}_1| = 2$ as there are 2 cache blocks in UCB_1. 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 the associated tasks, $\text{UCB}^{\text{G}} = \bigcup_{\forall \tau_i \in \Gamma^{\text{G}}} \text{UCB}_i$ and $\text{ECB}^{\text{G}} = \bigcup_{\forall \tau_i \in \Gamma^{\text{G}}} \text{ECB}_i$.

Each time a cache block is reloaded, a cost is introduced that is equal to the *block reload time* (BRT).

We assume a single level instruction only direct mapped cache. Data caches with a write-through policy are also supported however, further extension would be required for data caches under a write-back policy. In the case of set-associative *Least Recently Used* (LRU)[1] caches, a single cache-set may contain several UCBs. For example, UCB_1= {2,2,4} means that task τ_1 has two UCBs in cache-set 2 and one UCB in cache set 4. As one ECB suffices to evict all UCBs of the same cache-set, multiple accesses to the same set by the pre-empting task do not appear in the set of ECBs. A bound on the CRPD in the case of LRU caches due to task τ_j directly pre-empting τ_i is thus given by the intersection $\text{UCB}_i \cap' \text{ECB}_j = \{m | m \in \text{UCB}_i : m \in \text{ECB}_i\}$, where the result is a multiset that contains each element from UCB_i if it is also in ECB_j. A precise computation of CRPD in the case of LRU caches is given in Altmeyer et al. (2010). The equations provided in this paper can be applied to set-associative LRU caches with the above adaptation to the set-intersection.

3 Existing schedulability and CRPD analysis for FP scheduling

In this section we briefly recap how CRPD can be calculated in a single-level system scheduling using fixed priorities. Schedulability tests are used to determine if a taskset is schedulable, i.e. all the tasks will meet their deadlines given the worst-case pattern of arrivals and execution. For a given taskset, the response time R_i for each task τ_i, can be calculated and compared against the tasks' deadline, D_i. If every task in the taskset meets its deadline, then the taskset is schedulable. In the case of a single-level system, the equation used to calculate R_i is Audsley et al. (1993):

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

Equation (1) can be solved using fixed point iteration. Iteration continues until either $R_i^{\alpha+1} > D_i - J_i$ in which case the task is unschedulable, or until $R_i^{\alpha+1} = R_i^{\alpha}$ in which case the task is schedulable and has a worst-case response time of R_i^{α}. Note the convergence of (1) may be sped up using the techniques described in Davis et al. (2008).

To account for the CRPD, a term $\gamma_{i,j}$ is introduced into (1). There are a number of approaches that can be used, and for explanations of the analysis, see Altmeyer et al. (2012). In this work, we use the Combined Multiset approach by Altmeyer et al. (2012) for calculating the CRPD at task level. In this approach, $\gamma_{i,j}$ represents the total cost of all pre-emptions due to jobs of task τ_j executing within the response time of task τ_i. Incorporating $\gamma_{i,j}$ into (1) gives a revised equation for R_i:

[1] The concept of UCBs and ECBs cannot be applied to the FIFO or Pesudo-LRU replacement policies as shown by Burguiere et al. (2009).

$$R_i^{\alpha+1} = C_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^\alpha + J_j}{T_j} \right\rceil C_j + \gamma_{i,j} \right) \tag{2}$$

4 Schedulability analysis for hierarchical systems

Hierarchical scheduling is a technique that allows multiple independent components to be scheduled on the same system. A global scheduler allocates processing resources to each component via server capacity. Each component can then utilise the server capacity by scheduling its tasks using a local scheduler. A global scheduler can either be non-pre-emptive or pre-emptive. In this work we assume a non-pre-emptive global scheduler.

4.1 Supply bound function

In hierarchical systems, components do not have dedicated access to the processor, but must instead share it with other components. The *supply bound function* (Shin and Lee 2013), 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.

Figure 1 shows an example for server S^G with $Q^G = 5$ and $P^G = 8$. Here we assume the worst case scenario where a task is activated just after the server's budget is exhausted. In this case, the first instance of time at which tasks can receive some supply is at $2(P^G - Q^G) = 6$.

We define the *inverse supply bound function*, *isbf*, which gives the maximum amount of time needed by server S^G to supply some capacity c as $isbf^G$ (Richter 2005):

$$isbf^G(c) = c + (P^G - Q^G)\left(\left\lceil \frac{c}{Q^G} \right\rceil + 1\right) \tag{3}$$

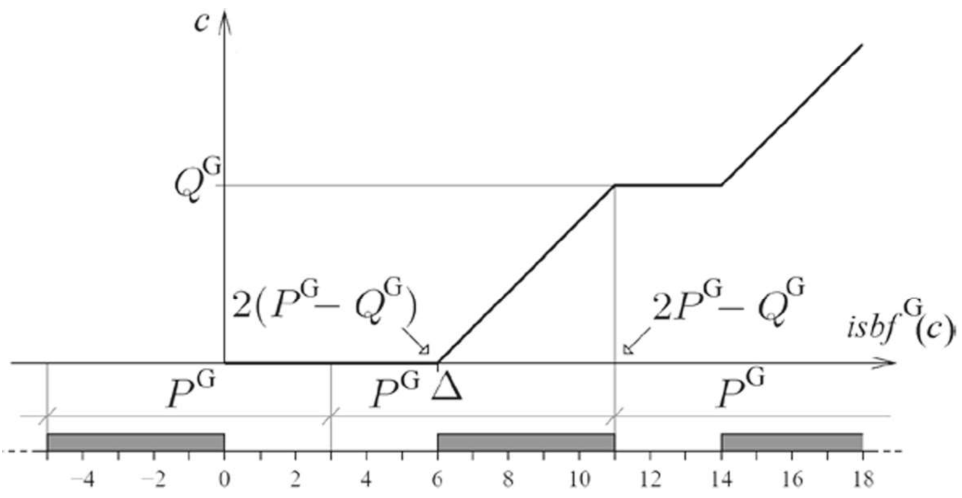


Fig. 1 General case of a server where $Q^G = 5$ and $P^G = 8$ showing it can take up to 6 time units before a task receives supply

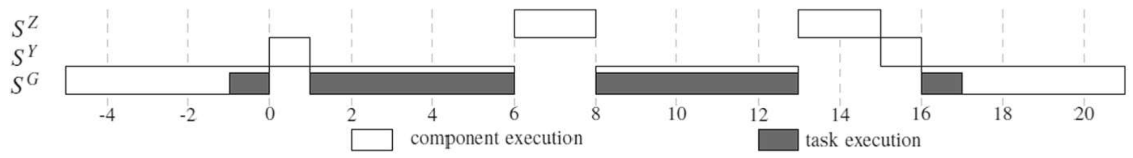


Fig. 2 Example global schedule to illustrate the server suspend and resume calculation with $P^G = P^Z = P^Y = 8$, $Q^G = 5$, $Q^Z = 2$, $Q^Y = 1$

In order to account for component level CRPD, we must define two terms. We use $E^G(t)$ to denote the maximum number of times server S^G can be both suspended and resumed within an internal of length t:

$$E^G(t) = 1 + \left\lfloor \frac{t}{P^G} \right\rfloor \tag{4}$$

Figure 2 shows an example global schedule for three components, G, Z and Y. When $t > 0$, server S^G can be suspended and resumed at least once. Then for each increase in t by P^G, server S^G could be suspended and resumed one additional time per increase in t by P^G. We note that this is a conservative bound on the number of times that a server is both suspended and resumed within an interval of length t.

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 may have loaded and need to reload. Note that if server S^Z runs more than once while server S^G is suspended, its tasks cannot evict the same blocks twice. As such, the number of disruptive executions is bounded by the number of times that server S^G can be both suspended and resumed. We use X^Z to denote the maximum number of such disruptive executions.

$$X^Z(S^G, t) = \min\left(E^G(t),\ 1 + \left\lceil \frac{t}{P^Z} \right\rceil\right) \tag{5}$$

Figure 3 shows an example global schedule for components G and Z. Between $t = 0$ and $t = 6$, component Z executes twice, but can only evict cache blocks that tasks in component G might have loaded and need to reload once.

Note that if multiple components e.g. Y and Z run while component G is suspended, then they can only evict the UCBs of component G once. The above observations are used in the derivation of CRPD analysis in the next section.

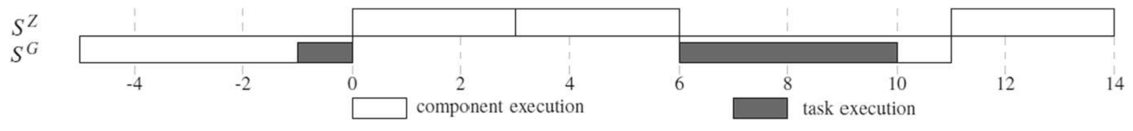


Fig. 3 Example global schedule to illustrate the disruptive execution calculation with $P^G = P^Z = 8$, $Q^G = 5$, $Q^Z = 3$

5 CRPD analysis for hierarchical systems with local FP scheduler

In this section, we describe how CRPD analysis can be extended for use in hierarchical systems with a local FP scheduler and integrated into the schedulability analysis for it. We do so by extending the concepts of ECB-only, UCB-only, UCB-union and UCB-union multiset analysis introduced in Busquets-Mataix et al. (1996), Lee et al. (1998), Tan and Mooney (2007) and Altmeyer et al. (2012) respectively to hierarchical systems. This analysis assumes a non-pre-emptive global scheduler such that the capacity of a server is supplied without pre-emption, but may be supplied starting at any time during the server's period. It assumes that tasks are scheduled locally using a pre-emptive fixed priority scheduler. We explain a number of different methods, building up in complexity.

The analysis needs to capture the cost of reloading any UCBs into cache that may be evicted by tasks belonging to other components; in addition to the cost of reloading any UCBs into cache that may be evicted by tasks in the same component. For calculating the intra-component CRPD, we use the Combined Multiset approach by Altmeyer et al. (2012). This can be achieved by combining the intra-component CRPD due to pre-emptions between tasks within the same component via the Combined Multiset approach, (2), with $isbf^G$, (3), with a new term, γ_i^G:

$$R_i^{\alpha+1} = isbf^G\left(C_i + \sum_{\forall j \in hp(G,i)} \left(\left\lceil \frac{R_i^\alpha + J_j}{T_j} \right\rceil C_j + \gamma_{i,j}\right) + \gamma_i^G\right) \tag{6}$$

Here, γ_i^G represents the CRPD on task τ_i in component G caused by tasks in the other components running while the server, S^G, for component G is suspended. Use of the inverse supply bound function gives the response time of τ_i under server, S^G, taking into account the shared access to the processor.

5.1 ECB-only

A simple approach to calculate component CPRD is to consider the maximum effect of the other components by assuming that every block evicted by the tasks in the other components has to be reloaded. There are two different ways to calculate this cost.

5.1.1 ECB-only-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 number of blocks that could be evicted. We then sum them up $E^G(R_i)$ times, where $E^G(R_i)$ upper bounds the number of times server S^G could be both suspended and resumed during the response time of task τ_i, see (4). We can calculate the CRPD impacting task τ_i of component G due to the other components in the system as:

$$\gamma_i^G = \mathrm{BRT} \cdot E^G(R_i) \cdot \left| \bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \mathrm{ECB}^Z \right| \tag{7}$$

5.1.2 ECB-only-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. This is achieved by calculating the number of disruptive executions that server S^Z can have on task τ_i in component G during the response time of task τ_i, given by $X^Z(S^G, R_i)$, see (5). We can then calculate an alternative bound for the CRPD incurred by task τ_i of component G due to the other components in the system as:

$$\gamma_i^G = \mathrm{BRT} \cdot \sum_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(X^Z(S^G, R_i) \cdot \left| \mathrm{ECB}^Z \right| \right) \tag{8}$$

Note that the ECB-only-all and ECB-only-counted approaches are incomparable.

5.2 UCB-only

Alternatively we can focus on the tasks in component G, hence calculating which UCBs could be evicted if the entire cache was flushed by the other components in the system. However, task τ_i may have been pre-empted by higher priority tasks. So we must bound the pre-emption cost by considering the number of UCBs over all tasks in component G that may pre-empt task τ_i, and task τ_i itself, given by $\tau_k \in \mathrm{hep}(G, i)$.

$$\bigcup_{\forall k \in \mathrm{hep}(G,i)} \mathrm{UCB}_k \tag{9}$$

We multiply the number of UCBs, (9), by the number of times that server S^G can be both suspended and resumed during the response time of task τ_i to give:

$$\gamma_i^G = \mathrm{BRT} \cdot E^G(R_i) \cdot \left| \bigcup_{\forall k \in \mathrm{hep}(G,i)} \mathrm{UCB}_k \right| \tag{10}$$

This approach is incomparable with the ECB-only-all and ECB-only-counted approaches.

5.3 UCB–ECB

While it is sound to only consider the ECBs of the tasks in the other components, or only the UCBs of the tasks in the component of interest, these approaches are clearly pessimistic. We can tighten the analysis by considering both.

5.3.1 UCB–ECB-all

We build upon the ECB-only-all and UCB-only methods. For task τ_i and all tasks that could pre-empt it in component G, we first calculate which UCBs could be evicted by the tasks in the other components, this is given by (9). We then take the union of all ECBs belonging to the other components to get the number of blocks that could potentially be evicted. We then calculate the intersection between the two unions to give an upper bound on the number of UCBs evicted by the ECBs of the tasks in the other components.

$$\left| \left(\bigcup_{\forall k \in \mathrm{hep}(G,i)} \mathrm{UCB}_k \right) \cap \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \mathrm{ECB}^Z \right) \right| \tag{11}$$

This is then multiplied by the number of times that the server S^G could be both suspended and resumed during the response time of task τ_i to give:

$$\gamma_i^G = \mathrm{BRT} \cdot E^G(R_i) \cdot \left| \left(\bigcup_{\forall k \in \mathrm{hep}(G,i)} \mathrm{UCB}_k \right) \cap \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \mathrm{ECB}^Z \right) \right| \tag{12}$$

By construction, the UCB–ECB-All approach dominates the ECB-only-all and UCB-only approaches.

5.3.2 UCB–ECB-counted

Alternatively, we can consider each component in isolation by building upon the ECB-only-counted and UCB-only approaches. For task τ_i and all tasks that could pre-empt it in component G, we start by calculating an upper bound on the number of blocks that could be evicted by component Z:

$$\left| \left(\bigcup_{\forall k \in \mathrm{hep}(G,i)} \mathrm{UCB}_k \right) \cap \mathrm{ECB}^Z \right| \tag{13}$$

We then multiply this number of blocks by the number of disruptive executions that server S^Z can have during the response time of task τ_i, and sum this up for all components to give:

$$\gamma_i^G = \mathrm{BRT} \cdot \sum_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(X^Z(S^G, R_i) \cdot \left| \left(\bigcup_{\forall k \in \mathrm{hep}(G,i)} \mathrm{UCB}_k \right) \cap \mathrm{ECB}^Z \right| \right) \tag{14}$$

By construction, the UCB–ECB-counted approach dominates the ECB-only-counted approach, but is incomparable with the UCB-only approach.

5.4 UCB–ECB-multiset

The UCB–ECB approaches are pessimistic in that they assume that each component can, directly or indirectly, evict UCBs of each task $\tau_k \in \mathrm{hep}(G, i)$ in component G up to$E^G(R_i)$ times during the response time of task τ_i. While this is potentially true when $\tau_k = \tau_i$, it can be a pessimistic assumption in the case of intermediate tasks which may have much shorter response times. The UCB–ECB-multiset approaches, described below, remove this source of pessimism by upper bounding the number of times intermediate task $\tau_k \in \mathrm{hep}(G, i)$ can run during the response time of τ_i. They then multiply this value by the number of times that the server S^G can be both suspended and resumed during the response time of task τ_k, $E^G(R_k)$.

5.4.1 UCB–ECB-multiset-all

First we form a multiset $M_{G,i}^{ucb}$ that contains the UCBs of task τ_k repeated $E^G(R_k)E_k(R_i)$ times for each task $\tau_k \in \mathrm{hep}(G, i)$. This multiset reflects the fact that the UCBs of task τ_k can only be evicted and reloaded $E^G(R_k)E_k(R_i)$ times during the response time of task τ_i as a result of server S^G being suspended and resumed.

$$M_{G,i}^{ucb} = \bigcup_{\forall k \in \mathrm{hep}(G,i)} \left(\bigcup_{E^G(R_k)E_k(R_i)} \mathrm{UCB}_k \right) \tag{15}$$

Then we form a second multiset $M_{G,i}^{ecb-A}$ that contains $E^G(R_i)$ 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 may subsequently need to be reloaded at most $E^G(R_i)$ times within the response time of task τ_i.

$$M_{G,i}^{ecb-A} = \bigcup_{E^G(R_i)} \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \mathrm{ECB}^Z \right) \tag{16}$$

The total CRPD incurred by task τ_i, in component G due to the other components in the system is then bounded by the size of the multiset intersection of $M_{G,i}^{ucb}$, (15), and

$M_{G,i}^{ecb-A}$, (16).

$$\gamma_i^G = \mathrm{BRT} \cdot \left| M_{G,i}^{ucb} \cap M_{G,i}^{ecb-A} \right| \tag{17}$$

5.4.2 UCB–ECB-multiset-counted

For the UCB–ECB-multiset-counted approach, we keep equation (15) for calculating the set of UCBs; however, we form a second multiset $M_{G,i}^{ecb-C}$ that contains $X^Z(S^G, R_i)$ copies of the ECBs of each other component Z in the system. This multiset reflects the fact that tasks of each server S^Z can evict blocks at most $X^Z(S^G, R_i)$ times within the response time of task τ_i.

$$M_{G,i}^{ecb-C} = \bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(\bigcup_{X^Z(S^G, R_i)} \mathrm{ECB}^Z \right) \tag{18}$$

The total CRPD incurred by task τ_i, in component G due to the other components in the system is then bounded by the size of the multiset intersection of $M_{G,i}^{ucb}$, (15), and $M_{G,i}^{ecb-C}$, (18).

$$\gamma_i^G = \mathrm{BRT} \cdot \left| M_{G,i}^{ucb} \cap M_{G,i}^{ecb-C} \right| \tag{19}$$

5.4.3 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. In both of these cases, only the UCB-only approach can be used as it requires no knowledge of the other components. We therefore present a variation called UCB–ECB-multiset-open that improves on UCB-Only while bounding the maximum component CRPD that could be caused by other unknown components. This approach draws on the benefits of the Multiset approaches, by counting the number of intermediate pre-emptions, while also recognising the fact that the cache utilisation of the other components can often be greater than the size of the cache. As such, the precise number of ECBs does not matter.

For the UCB–ECB-multiset-open approach we keep Eq. (15) for calculating the set of UCBs. Furthermore, we form a second multiset $M_{G,i}^{ecb-O}$ that contains $E^G(R_i)$ copies of all cache blocks. This multiset reflects the fact that server S^G can be both suspended and resumed, and the entire contents of the cache evicted at most $E^G(R_i)$ times within the response time of task τ_i.

$$M_{G,i}^{ecb-O} = \bigcup_{E^G(R_i)} (\{1, 2, \ldots N\}), \tag{20}$$

where N is the number of cache sets.

The total CRPD incurred by task τ_i, in component G due to the other unknown components in the system is then bounded by the size of the multiset intersection of $M_{G,i}^{ucb}$, (15), and $M_{G,i}^{ecb-O}$, (20).

$$\gamma_i^G = \text{BRT} \cdot \left| M_{G,i}^{ucb} \cap M_{G,i}^{ecb-O} \right| \tag{21}$$

5.5 Comparison of approaches

We have presented a number of approaches that calculate the CRPD due to global context switches, server switching, in a hierarchical system. All of the approaches can be applied to a system where full knowledge of all of the components is available i.e. a closed system. Two of the approaches, UCB-only and UCB–ECB-multiset-open, can also be applied to systems where there is no knowledge about the other components in the system i.e. these methods are applicable to open systems. This allows an upper bound on the inter-component CRPD to be calculated for a component in isolation.

Figure 4 shows a Venn diagram representing the relationships between the different approaches. The larger the area, the more tasksets the approach deems schedulable. The diagram highlights the incomparability between the '-All' and '-Counted' approaches, which results in them determining a different set of tasksets schedulable. The diagram also highlights dominance. For example, by construction, UCB–ECB-multiset-all dominates UCB–ECB-multiset-open as it also considers the tasks in the other components. Similarly, UCB–ECB-all dominates ECB-only-all as it considers

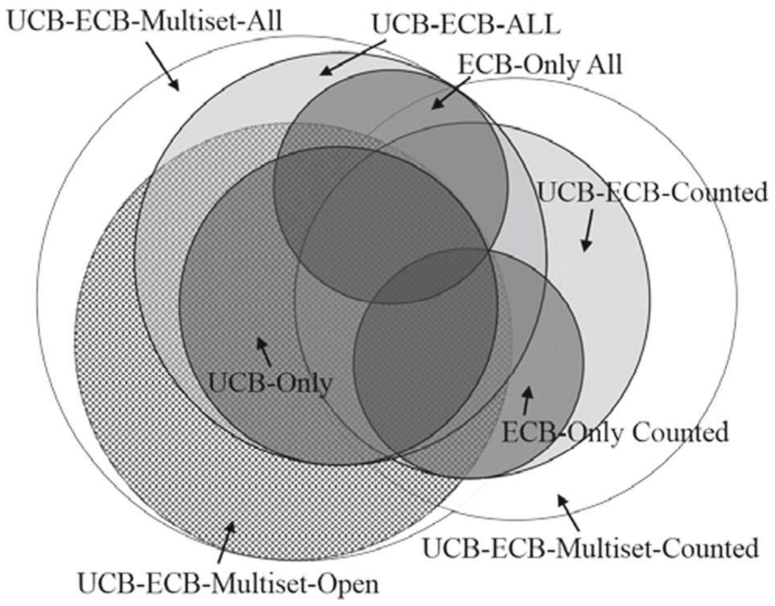


Fig. 4 Venn diagram showing the relationship between the different approaches

the tasks in all components whereas ECB-only-all does not consider the tasks in the suspended component.

We now give worked examples illustrating both incomparability and dominance relationships between the different approaches.

Consider the following example with three components, G, A and B, where component G has one task, Let BRT=1, $E^G(R_1) = 10$, $X^A(S^G, R_1) = 10$, $X^B(S^G, R_1) = 2$, $ECB^A = \{1, 2\}$ and $ECB^B = \{3, 4, 5, 6, 7, 8, 9, 10\}$. In this example components A and G run at the same rate, while component B runs at a tenth of the rate of component G.

ECB-only-all considers the ECBs of component B effectively assuming that component B runs at the same rate as component G:

$$\begin{aligned}
\gamma_1^G &= \text{BRT} \times E^G(R_1) \times \left|\text{ECB}^A \cup \text{ECB}^B\right| \\
\gamma_1^G &= 1 \times 10 \times |\{1, 2\} \cup \{3, 4, 5, 6, 7, 8, 9, 10\}| \\
&= 10 \times |\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}| = 10 \times 10 = 100
\end{aligned}$$

By comparison ECB-only-counted considers components A and B individually, and accounts for the ECBs of component B based on the number of disruptive executions that it may have.

$$\begin{aligned}
\gamma_1^G &= \text{BRT} \times \left(X^A(S^G, R_1) \times \left|\text{ECB}^A\right| + X^B(S^G, R_1) \times \left|\text{ECB}^B\right|\right) \\
\gamma_1^G &= 1 \times (10 \times |\{1, 2\}| + 2 \times |\{3, 4, 5, 6, 7, 8, 9, 10\}|) \\
&= (10 \times 2) + (2 \times 8) = 36
\end{aligned}$$

We now present a more detailed worked example for all approaches where the ECB-only-all approach outperforms the ECB-only-counted approach. This confirms the incomparability of the -All and -Counted approaches.

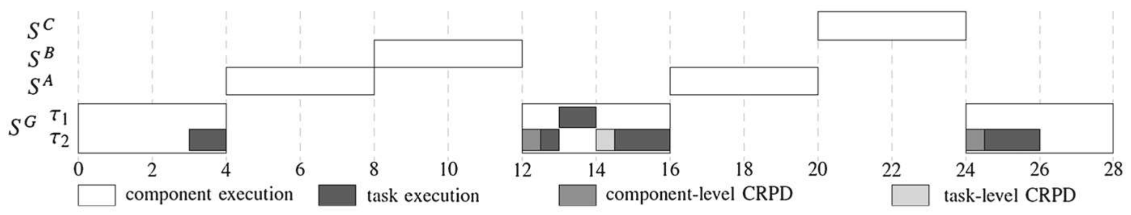
Figure 5 shows an example schedule for four components, G, A, B and C, where component G has two tasks. Let BRT=1, $E^G(R_1) = 1$, $E^G(R_2) = 2$, $E_1(R_2) = 1$ and $E_2(R_2) = 1$, and the number of disruptive executions be:

$X^A(S^G, R_1) = 1$, $X^B(S^G, R_1) = 1$, $X^C(S^G, R_1) = 1$ and $X^A(S^G, R_2) = 2$, $X^B(S^G, R_2) = 2$, $X^C(S^G, R_2) = 2$.

The following examples show how some of the approaches calculate the component CRPD for task τ_2 of component G.

ECB-only-all:

$$\begin{aligned}
\gamma_2^G &= \text{BRT} \times E^G(R_2) \times \left|\text{ECB}^A \cup \text{ECB}^B \cup \text{ECB}^C\right| \\
\gamma_2^G &= 1 \times 2 \times |\{2, 3, 4, 5, 6, 7, 8\} \cup \{2, 3, 4, 5\} \cup \{4, 5, 6, 7, 8, 9, 10\}| \\
&= 2 \times |\{2, 3, 4, 5, 6, 7, 8, 9, 10\}| = 2 \times 9 = 18
\end{aligned}$$



	UCB	ECB
S^C	-	{4,5,6,7,8,9,10}
S^B	-	{2,3,4,5}
S^A	-	{2,3,4,5,6,7,8}
τ_1	{2}	{2,3}
τ_2	{1,2,3}	{1,2,3,4}

Fig. 5 Example schedule and UCB/ECB data for four components to demonstrate how the different approaches calculate CRPD

ECB-only-counted:

$$\begin{aligned}\gamma_2^G &= \text{BRT}\times\Big(X^A(S^G,R_2)\times\Big|\text{ECB}^A\Big|+X^B(S^G,R_2)\times\Big|\text{ECB}^B\Big|+X^C(S^G,R_2)\times\Big|\text{ECB}^C\Big|\Big)\\ \gamma_2^G &= 1\times(2\times|\{2,3,4,5,6,7,8\}|+2\times|\{2,3,4,5\}|+2\times|\{4,5,6,7,8,9,10\}|)\\ &= (2\times 7)+(2\times 4)+(2\times 7)=36\end{aligned}$$

UCB-only:

$$\begin{aligned}\gamma_2^G &= \text{BRT}\times E^G(R_2)\times|\text{UCB}_1\cup\text{UCB}_2|\\ \gamma_2^G &= 1\times 2\times(|\{2\}\cup\{1,2,3\}|)\\ &= 2\times|\{1,2,3\}|=6\end{aligned}$$

All of those approaches overestimated the CRPD, although UCB-only achieves a much tighter bound than the ECB-only-all and ECB-only-counted approaches. The bound can be tightened further by using the more sophisticated approaches, for example, UCB–ECB-multiset-counted:

$$\begin{aligned}M_{G,2}^{ucb} &= \left(\bigcup_{E^G(R_1)E_1(R_2)}\text{UCB}_1\right)\cup\left(\bigcup_{E^G(R_2)E_2(R_2)}\text{UCB}_2\right)\\ M_{G,2}^{ucb} &= \{2\}\cup\{1,2,3\}\cup\{1,2,3\}=\{1,1,2,2,2,3,3\}\end{aligned}$$

$$M_{G,2}^{ecb-C}=\left(\bigcup_{X^A(S^G,R_2)}\text{ECB}^A\right)\cup\left(\bigcup_{X^B(S^G,R_2)}\text{ECB}^B\right)\cup\left(\bigcup_{X^C(S^G,R_2)}\text{ECB}^C\right)$$

$$
\begin{aligned}
M_{G,2}^{ecb-C} &= \{2,3,4,5,6,7,8\} \cup \{2,3,4,5,6,7,8\} \cup \{2,3,4,5\} \cup \{2,3,4,5\} \\
&\quad \cup \{4,5,6,7,8,9,10\} \cup \{4,5,6,7,8,9,10\} \\
&= \{2,2,2,2,3,3,3,3,4,4,4,4,4,4,5,5,5,5,5,5,6,6,6,6,7,7, \\
&\quad 7,7,,8,8,8,8,9,9,10,10\}
\end{aligned}
$$

$$
\gamma_2^G = \text{BRT} \times \left| M_{G,2}^{ucb} \cap M_{G,2}^{ecb-C} \right| = 1 \times |\{2,2,2,3,3\}| = 5
$$

In this specific case, the UCB–ECB-multiset-all approach calculates the tightest bound:

$$
\begin{aligned}
M_{G,2}^{ecb-A} &= \bigcup_{E^G(R_2)} \left(\text{ECB}^A \cup \text{ECB}^B \cup \text{ECB}^C \right) \\
M_{G,2}^{ecb-A} &= \bigcup_{2} (\{2,3,4,5,6,7,8\} \cup \{2,3,4,5\} \cup \{4,5,6,7,8,9,10\}) \\
&= \bigcup_{2} (\{2,3,4,5,6,7,8,9,10\}) \\
&= \{2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10\} \\
\gamma_2^G &= \text{BRT} \times \left| M_{G,2}^{ucb} \cap M_{G,2}^{ecb-A} \right| = 1 \times |\{2,2,3,3\}| = 4
\end{aligned}
$$

Assuming there are 12 cache sets in total,[2] the UCB–ECB-multiset-open approach gives:

$$
\begin{aligned}
M_{G,2}^{ecb-O} &= \bigcup_{E^G(R_2)} (\{1,2,3,4,5,6,7,8,9,10,11,12\}) \\
M_{G,2}^{ecb-O} &= \bigcup_{2} (\{1,2,3,4,5,6,7,8,9,10,11,12\}) \\
&= \{1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,11,11,12,12\} \\
\gamma_2^G &= \text{BRT} \times \left| M_{G,2}^{ucb} \cap M_{G,2}^{ecb-O} \right| = 1 \times |\{1,1,2,2,3,3\}| = 6
\end{aligned}
$$

6 Existing schedulability and CRPD analysis for EDF scheduling

In this section we briefly recap how CRPD can be calculated in a single-level system scheduled using EDF.

EDF is a dynamic scheduling algorithm which always schedules the job with the earliest absolute deadline first. In pre-emptive EDF, any time a job arrives with an earlier absolute deadline than the current running job, it will pre-empt the current job.

[2] Although we used 12 cache sets in this example, we note that the result obtained is in fact independent of the total number of cache sets.

When a job completes its execution, the EDF scheduler chooses the pending job with the earliest absolute deadline to execute next.

Liu and Layland (1973) gave a necessary and sufficient schedulability test that indicates whether a taskset is schedulable under EDF iff $U \leq 1$, under the assumption that all tasks have implicit deadlines (D_i=T_i). In the case where $D_i \neq T_i$ this test is still necessary, but is no longer sufficient.

Dertouzos (1974) proved EDF to be optimal among all scheduling algorithms on a uniprocessor, in the sense that if a taskset cannot be scheduled by pre-emptive EDF, then this taskset cannot be scheduled by any algorithm.

Leung and Merrill (1980) showed that a set of periodic tasks is schedulable under EDF iff all absolute deadlines in the interval [0,max$\{s_i\}$+2H] are met, where s_i is the start time of task τ_i, min$\{s_i\}$=0, and H is the hyperperiod (least common multiple) of all tasks' periods.

Baruah et al. (1990a, b) extended Leung and Merrill's work (1980) to sporadic tasksets. They introduced $h(t)$, the *processor demand function*, which denotes 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. Using this they showed that a taskset is schedulable iff $\forall t > 0,\ h(t) \leq t$ where $h(t)$ is defined as:

$$h(t) = \sum_{i=1}^{n} \max\left\{0,\, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right\} C_i \tag{22}$$

Examining (22), it can be seen that $h(t)$ can only change when t is equal to an absolute deadline, which restricts the number of values of t that need to be checked. In order to place an upper bound on t, and therefore the number of calculations of $h(t)$, the minimum interval in which it can be guaranteed that an unschedulable taskset will be shown to be unschedulable must be found. For a general taskset with arbitrary deadlines t can be bounded by L_a (George et al. 1996):

$$L_a = \max\left\{D_1, \ldots, D_n, \frac{\sum_{i=1}^{n} (T_i - D_i)\, U_i}{1 - U}\right\} \tag{23}$$

Spuri (1996) and Ripoll et al. (1996) showed that an alternative bound L_b, given by the length of the synchronous busy period can be used. L_b is computed by solving the following equation using fixed point iteration:

$$w^{\alpha+1} = \sum_{i=1}^{n} \left\lceil \frac{w^{\alpha}}{T_i} \right\rceil C_i \tag{24}$$

There is no direct relationship between L_a and L_b, which enables t to be bounded by L= min(L_a, L_b). Combined with the knowledge that $h(t)$ can only change at an absolute deadline, a taskset is therefore schedulable under EDF iff $U \leq 1$ and:

$$\forall t \in Q, h(t) \leq t \tag{25}$$

where Q is defined as:

$$Q = \{d_k | d_k = kT_i + D_i \wedge d_k < \min(L_a, L_b), k \in N\} \tag{26}$$

Zhang and Burns (2009) presented their Q*uick convergence Processor-demand Analysis* (QPA) algorithm which exploits the monotonicity of $h(t)$. QPA determines schedulability by starting with a value of t that is close to L, and then iterating back towards 0 checking a significantly smaller number of values of t than would otherwise be required.

Task level CRPD analysis can be integrated into the EDF schedulability test by introducing an additional parameter, $\gamma_{t,j}$ (Lunniss et al. 2013). In this paper we use the Combined Multiset approach by Lunniss et al. (2013) where $\gamma_{t,j}$ represents the cost of the maximum number $E_j(t)$ of pre-emptions by jobs of task τ_j that have their release times and absolute deadlines in an interval of length t. It is therefore included in (22) as follows:

$$h(t) = \sum_{j=1}^{n} \left(\max\left\{0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right\} C_j + \gamma_{t,j} \right) \tag{27}$$

$\gamma_{t,j}$ can then be calculated using two different methods and the lowest value of the two used to calculate the processor demand. These methods calculate the cost of each possible individual pre-emption by task τ_j that could occur during an interval of length t.

7 CRPD analysis for hierarchical systems with local EDF scheduler

In this section, we present CRPD analysis for hierarchical systems with a local EDF scheduler by adapting the analysis that we presented for a local FP scheduler in Sect. 5.

Overall, the analysis must account for the cost of reloading any UCBs into cache that may be evicted by tasks running in the other components. This is in addition to the cost of reloading any UCBs into cache that may be evicted by tasks in the same component. For calculating the intra-component CRPD, we use the Combined Multiset approach by Lunniss et al. (2013) for EDF scheduling of a single level system. 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 (27) with $isbf^G$, (3), and γ_t^G, we get the following expression for the modified processor demand[3] within an interval of length t:

$$h(t) = isbf^G \left(\sum_{j=1}^{n} \left(\max\left\{0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right\} C_j + \gamma_{t,j} \right) + \gamma_t^G \right) \tag{28}$$

[3] Strictly, $h(t)$ is the maximum time required for the server to provide the processing time demand.

In order to account for component CRPD we must define an additional term. The set of tasks in component G that can be affected by the server S^G being both suspended and resumed in an interval of length t, aff(G,t) is based on the relative deadlines of the tasks. It captures all of the tasks whose relative deadlines are less than or equal to t as they need to be included when calculating $h(t)$. This gives:

$$\mathit{aff}(G,t) = \left\{\tau_i \in \Gamma^G | t \geq D_i\right\} \tag{29}$$

The primary difference between the analysis for FP and EDF, is that under EDF we must determine the CRPD within an interval of length t, rather than within the response time of a task. The other notable difference is that because the priorities of tasks are dynamic, we must use the set of tasks that could be affected by server S^G being both suspended and resumed in an interval of length t, rather than using the fixed set of tasks with higher priorities. Therefore, the approaches presented for FP scheduling can be adapted for EDF by making the following changes:

- Replace R_i with t
- Replace hep(G,t) with aff(G,t)
- For the multiset approaches when considering intermediate tasks, replace R_k with D_k
- Use (28) to determine schedulability

We now list the revised equations for determining the inter-component CRPD under EDF.
ECB- ONLY- ALL

$$\gamma_t^G = \text{BRT} \cdot E^G(t) \cdot \left| \bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \text{ECB}^Z \right| \tag{30}$$

ECB- ONLY- COUNTED

$$\gamma_t^G = \text{BRT} \cdot \sum_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(X^Z(S^G, t) \cdot \left| \text{ECB}^Z \right| \right) \tag{31}$$

UCB- ONLY

$$\gamma_t^G = \text{BRT} \cdot E^G(t) \cdot \left| \bigcup_{\forall k \in \text{aff}(G,t)} \text{UCB}_k \right| \tag{32}$$

UCB–ECB- ALL

$$\gamma_t^G = \mathrm{BRT} \cdot E^G(t) \cdot \left| \left(\bigcup_{\forall k \in \mathrm{aff}(G,t)} \mathrm{UCB}_k \right) \cap \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \mathrm{ECB}^Z \right) \right| \tag{33}$$

UCB–ECB- COUNTED

$$\gamma_t^G = \mathrm{BRT} \cdot \sum_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(X^Z(S^G, t) \cdot \left| \left(\bigcup_{\forall k \in \mathrm{aff}(G,t)} \mathrm{UCB}_k \right) \cap \mathrm{ECB}^Z \right| \right) \tag{34}$$

7.1 Multiset approaches

$$M_{G,t}^{ucb} = \bigcup_{\forall k \in \Gamma^G} \left(\bigcup_{E^G(D_k)E_k(t)} \mathrm{UCB}_k \right) \tag{35}$$

UCB–ECB- MULTISET- ALL

$$M_{G,t}^{ecb-A} = \bigcup_{E^G(t)} \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \mathrm{ECB}^Z \right) \tag{36}$$

$$\gamma_t^G = \mathrm{BRT} \cdot \left| M_{G,t}^{ucb} \cap M_{G,t}^{ecb-A} \right| \tag{37}$$

UCB–ECB- MULTISET- COUNTED

$$M_{G,t}^{ecb-C} = \bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(\bigcup_{X^Z(S^G,t)} \mathrm{ECB}^Z \right) \tag{38}$$

$$\gamma_t^G = \mathrm{BRT} \cdot \left| M_{G,t}^{ucb} \cap M_{G,t}^{ecb-C} \right| \tag{39}$$

UCB–ECB- MULTISET- OPEN

$$M_{G,t}^{ecb-O} = \bigcup_{E^G(t)} (\{1, 2, ..N\}) \tag{40}$$

$$\gamma_t^G = \mathrm{BRT} \cdot \left| M_{G,t}^{ucb} \cap M_{G,t}^{ecb-O} \right| \tag{41}$$

7.2 Effect on task utilisation and h(t) calculation

As the component level CRPD analysis effectively inflates the execution time of tasks by the CRPD that can be incurred in an interval of length t, the upper bound L, used for calculating the processor demand $h(t)$, must be adjusted. This is an extension to the adjustment that must be made for task level CRPD as described in Sect. V. D. "Effect on Task Utilisation and h(t) Calculation" in Lunniss et al. (2013). This is achieved by calculating an upper bound on the utilisation due to CRPD that is valid for all intervals of length greater than some value L_c. This CRPD utilisation value is then used to inflate the taskset utilisation and thus compute an upper bound L_d on the maximum length of the busy period. This upper bound is valid provided that it is greater than L_c, otherwise the actual maximum length of the busy period may lie somewhere in the interval $[L_d, L_c]$, hence we can use max(L_c, L_d) as a bound.

We assign $t = L_c$= 100 T_{max} to ensure that the maximum overestimation of both the task level CRPD utilisation $U^{\gamma} = \gamma_t/t$ and the component level CRPD utilisation $U^{\gamma G} = \gamma_t^G/t$ is 1 %, since the CRPD for at most one extra period may be included. We determine $U^{\gamma G}$ by calculating γ_t^G, however, when calculating the multiset of the UCBs that could be affected $M_{G,t}^{ucb}$, (35), $E_x^{max}(t)$ is substituted for $E_x(t)$ to ensure that the computed value of $U^{\gamma G}$ is a valid upper bound for all intervals of length $t \geq L_c$.

$$E_x^{max}(t) = \max\left(0, 1 + \left\lceil \frac{t - D_x}{T_x} \right\rceil\right) \tag{42}$$

We use a similar technique of substituting $E_x^{max}(t)$ for $E_x(t)$ in the calculation of the task level CRPD (as described in Sect. V. D. of Lunniss et al. 2013), to give U^{γ}.

If $U + U^{\gamma} + U^{\gamma G} \geq 1$, then the taskset is deemed unschedulable, otherwise an upper bound on the length of the busy period can be computed via a modified version of (25):

$$w^{\alpha+1} \leq \sum_{\forall j} \left(\frac{w^{\alpha}}{T_j} + 1 \right) C_j + w^{\alpha} U^{\gamma} \tag{43}$$

rearranged to give:

$$w \leq \frac{1}{(1 - (U + U^{\gamma} + U^{\gamma G}))} \sum_{\forall j} U_j T_j \tag{44}$$

Then, substituting in T_{max} for each value of T_j the upper bound is given by:

$$L_d = \frac{U \cdot T_{max}}{\left(1 - (U + U^{\gamma} + U^{\gamma G})\right)} \tag{45}$$

Finally, $L = max(L_c, L_d)$ can then be used as the maximum value of t to check in the EDF schedulability test.

7.3 Comparison of approaches

In this section we have presented a number of approaches for calculating component CRPD in a hierarchical system with a local EDF scheduler. These approaches all have the same dominance and incomparability relationships as the approaches presented in Sect. 5 for a local FP scheduler. We therefore refer the reader to Sect. 5.5 for an explanation of the relationships between the approaches. However, the relative performance between the approaches differs from the FP variants as shown in the next section.

8 Case study

In this section we compare the different approaches for calculating CRPD in hierarchical scheduling using tasksets based on a case study. The case study uses PapaBench[4] which is a real-time embedded benchmark based on the software of a GNU-license UAV, called Paparazzi. WCETs, UCBs, and ECBs were calculated for the set of tasks using aiT[5] based on 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. This configuration was chosen so as to give representative results when using the relatively small benchmarks that were available to us. WCETs, periods, UCBs, and ECBs for each task based on the target system are provided in Table 1. We made the following assumptions in our evaluation to handle the interrupt tasks:

- Interrupts have a higher priority than the servers and normal tasks.
- Interrupts cannot pre-empt each other.
- Interrupts can occur at any time.
- All interrupts have the same deadline which must be greater than or equal to the sum of their execution times in order for them to be schedulable.
- The cache is disabled whenever an interrupt is executing and enabled again after it completes.

Based on these assumptions, we integrated interrupts into the model by replacing the server capacity Q^G in Eq. (3) by $Q^G - I^G$, where I^G is the maximum execution time of all interrupts in an interval of length Q^G. This effectively assumes that the worst case arrival of interrupts could occur in any component and steals time from its budget.

We assigned a deadline of 2ms to all of the interrupt tasks, and implicit deadlines so that $D_i = T_i$, to the normal tasks. We then calculated the total utilisation for the system and then scaled T_i and D_i up for all tasks in order to reduce the total utilisation to the target utilisation for the system. We used the number of UCBs and ECBs obtained via analysis, placing the UCBs in a group at a random location in each task. We then generated 1000 systems each containing a different allocation of tasks to each component, using the following technique. We split the normal tasks at

[4] http://www.irit.fr/recherches/ARCHI/MARCH/rubrique.php3?id_rubrique=97.

[5] http://www.absint.com/ait/.

Table 1 Execution times, periods and number of UCBs and ECBs for the tasks from PapaBench

Task		UCBs	ECBs	WCET (ms)	Period (ms)
Fly-by-wire					
I1	interrupt_radio	2	10	0.210	25
I2	interrupt_servo	1	6	0.167	50
I3	interrupt_spi	2	10	0.256	25
T1	check_failsafe	10	132	1.240	50
T2	check_mega128_values	10	130	5.039	50
T3	send_data_to_autopilot	10	114	2.283	25
T4	servo_transmit	2	10	2.059	50
T5	test_ppm	30	255	12.579	25
Autopilot					
I4	interrupt_modem	2	10	0.303	100
I5	interrupt_spi_1	1	10	0.251	50
I6	interrupt_spi_2	1	4	0.151	50
I7	interrupt_gps	3	26	0.283	250
T5	altitude_control	20	66	1.478	250
T6	climb_control	1	210	5.429	250
T7	link_fbw_send	1	10	0.233	50
T8	navigation	10	256	4.432	250
T9	radio_control	0	256	15.681	25
T10	receive_gps_data	22	194	5.987	250
T11	reporting	2	256	12.222	100
T12	stabilization	11	194	5.681	50

random into 3 components with four tasks in two components and five in the other. In the case of local FP scheduling, we assigned task priorities according to deadline monotonic priority assignment (Liu and Layland 1973). Next we set the period of each component's server to 12.5 ms, which is half the minimum task period. Finally, we organised tasks in each component in memory in a sequential order based on their priority for FP, or their unique task index for EDF. Due to task index assignments, this gave the same task layout in both cases. We then ordered components in memory sequentially based on their index.

For each system the total task utilisation 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, the utilisation of all of its tasks. We then performed a binary search between this minimum and the maximum, 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. In addition to evaluating each of the presented approaches,

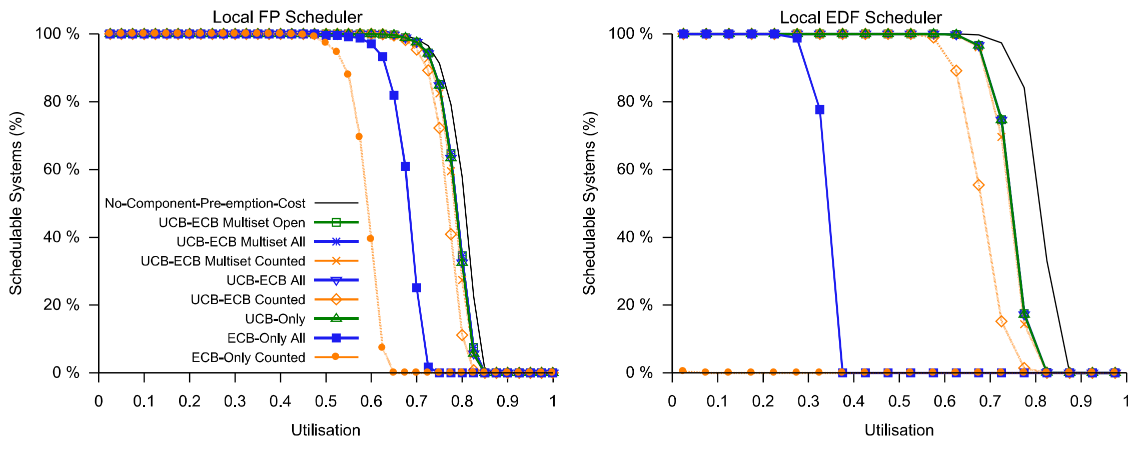


Fig. 6 Percentage of schedulable tasksets at each utilisation level for the case study tasksets

we also calculated schedulability based on no component pre-emption costs, but still including task level CRPD. For every approach the intra-component CRPD, between tasks in the same component, was calculated using either the Combined Multiset approach for FP (Altmeyer et al. 2012), or the Combined Multiset approach for EDF (Lunniss et al. 2013).

The results for the case study for a local FP scheduler and local EDF scheduler are shown in Fig. 6, note that graphs are best viewed online in colour. Although we generated 1000 systems, they were all very similar as they are made up of the same set of tasks. The first point to note is that the FP approaches deem a higher number of tasksets schedulable than the EDF ones, despite EDF having a higher number of schedulable tasksets for the no-component-pre-emption-cost case. In Sect. 9, we explore the source of pessimism in the EDF analysis. Focusing on the different approaches, ECB-only-counted and ECB-only-all perform the worst as they only consider the other components in the system. In the case of a local EDF scheduler, the ECB-only-counted approach is unable to deem any tasksets schedulable except at the lowest utilisation level. Next was UCB–ECB-counted which though it considers all components, accounts for the other components pessimistically in this case study, since all servers have the same period. The remainder of the approaches all had very similar performance.

We note that no-component-pre-emption-cost reveals that the pre-emption costs are very small for the PapaBench tasks. This is due to a number of factors including the nearly harmonic periods, small range of task periods, and relatively low number of ECBs for many tasks (Fig. 6).

9 Evaluation

In this section we compare the different approaches for calculating CRPD in hierarchical scheduling using synthetically generated tasksets. This allows us to explore a wider range of parameters and therefore give some insight into how the different approaches perform in a variety of cases.

To generate the components and tasksets we generated n, default of 24, tasks using the UUnifast algorithm (Bini and Buttazzo 2005) 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 generated two sets of tasksets, one with implicit deadlines, so that $D_i = T_i$, and one with constrained deadlines. We used $D_i = y + x(T_i - y)$ to generate the constrained deadlines, where x is a uniform random number between 0 and 1, and $y = max(T_i/2, 2C_i)$. This generates constrained deadlines that are no less than half the period of the tasks. All results presented in Sect. 9.1 are for tasks with implicit deadlines. In general the results for constrained deadlines were similar with a lower number of systems deemed schedulable. The exception to this is that under a local EDF scheduler, the UCB–ECB-multiset approaches showed an increase in schedulability when deadlines were reduced by a small amount. This behaviour is investigated and explained in Sect. 9.2.

We used the UUnifast algorithm to generate the number of ECBs for each task so that the ECBs added up to the desired cache utilisation, default of 10. 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. In the case of a local FP scheduler, we assigned task priorities according to Deadline Monotonic priority assignment. Next we set the period of each component's server to 5ms, which was half the minimum possible task period. Finally we organised tasks in each component in memory in a sequential order based on their priority for FP, or their unique task index for EDF, which gave the same task layout in both cases, and then ordered components in memory sequentially based on their index. We generated 1000 systems using this technique.

In our evaluations we used the same local scheduler in each component, so that all components were scheduled locally using either FP or EDF. However, we note that the analysis is not dependent on the scheduling policies of the other components and hence can be applied to a system where some components are scheduled locally using FP and others using EDF.

We determined the schedulability of the synthetic tasksets using the approach described in the fourth paragraph of Sect. 8.

9.1 Baseline evaluation

We investigated the effect of key cache and taskset configurations on the analysis by varying the following key parameters:

- Number of components (default of 3)
- Server period (default of 5ms)
- Cache Utilisation (default of 10)
- Total number of tasks (default of 24)
- Range of task periods (default of [10, 1000]ms)

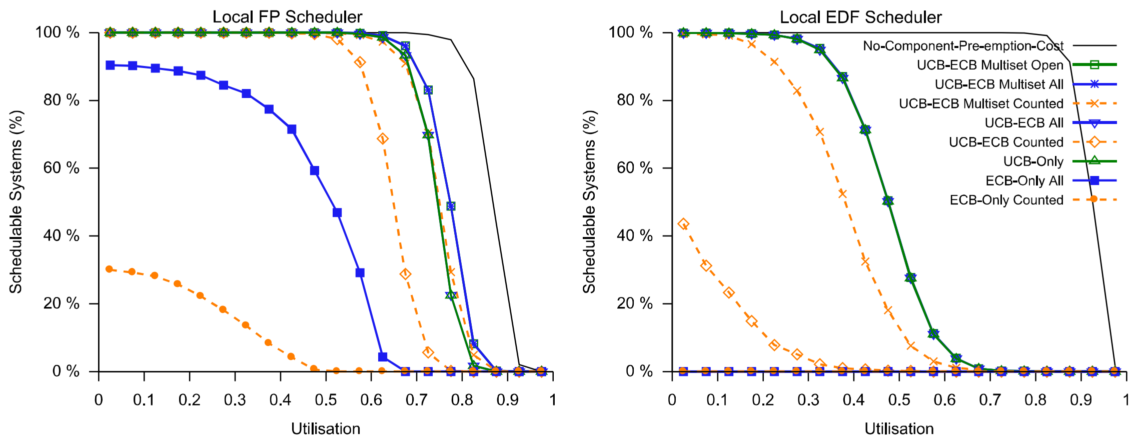


Fig. 7 Percentage of schedulable tasksets at each utilisation level for the synthetic tasksets

The results for the baseline evaluation under implicit deadline tasksets are shown in Fig. 7. The results again show that the analysis for determining inter-component CRPD for a local FP scheduler deems a higher number of systems schedulable than the analysis for a local EDF scheduler. In the case of a local EDF scheduler, both ECB-only approaches deemed no tasksets schedulable. In the case of a local FP scheduler ECB-only-counted is least effective as it only considers the other components and does so individually, followed by ECB-only-all. UCB–ECB-counted deemed a higher number of tasksets schedulable, although it deemed significantly fewer for a local EDF scheduler than with a local FP scheduler. Under EDF, UCB–ECB-multiset-counted was next, followed by all other approaches. Under FP, UCB–ECB-multiset-counted performed similarly to UCB-only and UCB–ECB-all, crossing over at a utilisation of 0.725 highlighting their incomparability. Although UCB–ECB-all dominates UCB-only, it can only improve over UCB-only when the cache utilisation of the other components is sufficiently low that they cannot evict all cache blocks. The UCB–ECB-multiset-all and UCB–ECB-multiset-open approaches performed the best for both types of local scheduler.

Despite only considering the properties of the component under analysis, the UCB–ECB-multiset-open approach proved highly effective. The reason for this is that once the size of the other components that can run while a given component is suspended is equal to or greater than the size of the cache then UCB–ECB-multiset-all and UCB–ECB-multiset-open become equivalent.

Consider the UCB–ECB-Multiset approaches under a local EDF scheduler. Examining Eq. (35), we note that $E^G(D_k)E_k(t)$ is based on the deadline of a task. Therefore, the analysis under implicit deadlines effectively assumes the UCBs of all tasks in component G could be in use each time the server for component G is suspended. Whereas, under a local FP scheduler the analysis is able to bound how many times the server for component G is suspended and resumed based on the computed response time of each task which for many tasks is much less than its deadline, and period. Figure 8 shows a subset of the results presented in Fig. 7. When component CRPD is not considered, EDF outperforms FP. However, once component CRPD is taken into account, the analysis for FP significantly outperforms the analysis for EDF.

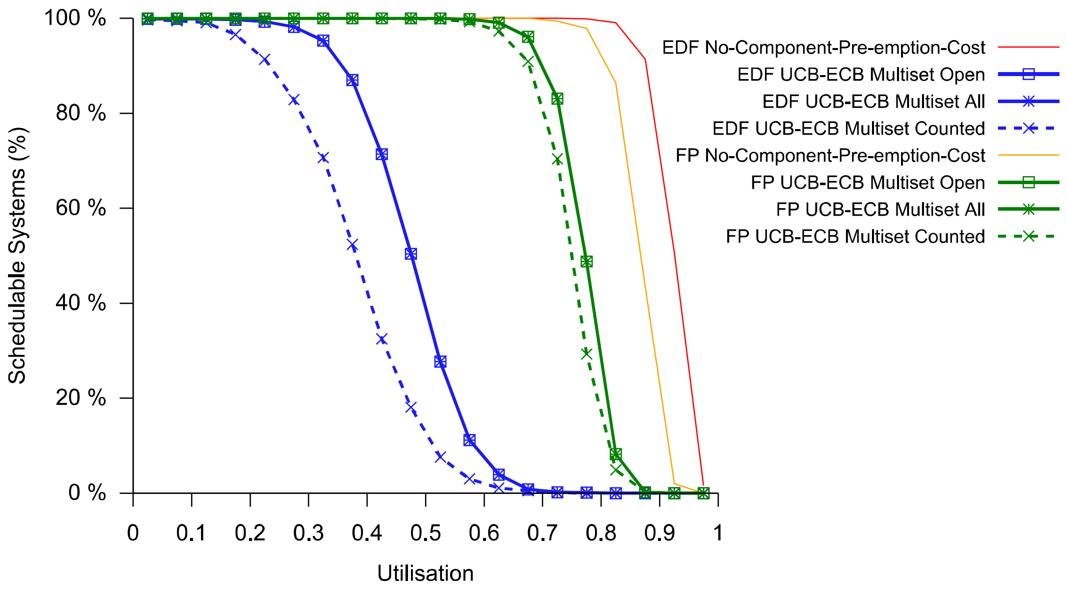


Fig. 8 Percentage of schedulable tasksets at each utilisation level for the synthetic tasksets directly comparing the analysis for local FP and EDF schedulers

9.2 Weighted schedulability

Evaluating all combinations of different parameters is not possible. Therefore, the majority of our evaluations focused on varying one parameter at a time. To present the results, weighted schedulability measures (Bastoni et al. 2010) are used. The benefit of using a weighted schedulability measure is that it reduces a 3-dimensional plot to 2 dimensions. Individual results are weighted by taskset utilisation to reflect the higher value placed on a being able to schedule higher utilisation tasksets. We used 100 systems for each utilisation level from 0.025 to 1.0 in steps of 0.025 for the weighted schedulability experiments.

9.2.1 Number of components

To investigate the effects of splitting the overall set of tasks into components, we fixed the total number of tasks in the system at 24, and then varied the number of components from 1, with 24 tasks in one component, to 24, with 1 task per component, see Fig. 9. Components were allocated an equal number of tasks where possible, otherwise tasks were allocated to each component in turn until all tasks where allocated. We note that with one component, the UCB-only and UCB–ECB-multiset-open approaches calculate a non-zero inter-component CRPD. This is because they assume that every time a component is suspended its UCBs are evicted, even though there is only one component running in the system. With two components the ECB-only-all and ECB-only-counted approaches are equal. Above two components the ECB-only-all, ECB-only-counted and UCB–ECB-counted approaches get rapidly worse as they over-count blocks. Under a local FP scheduler, all other approaches improve as the number of components is increased above 2 up to 8 components.

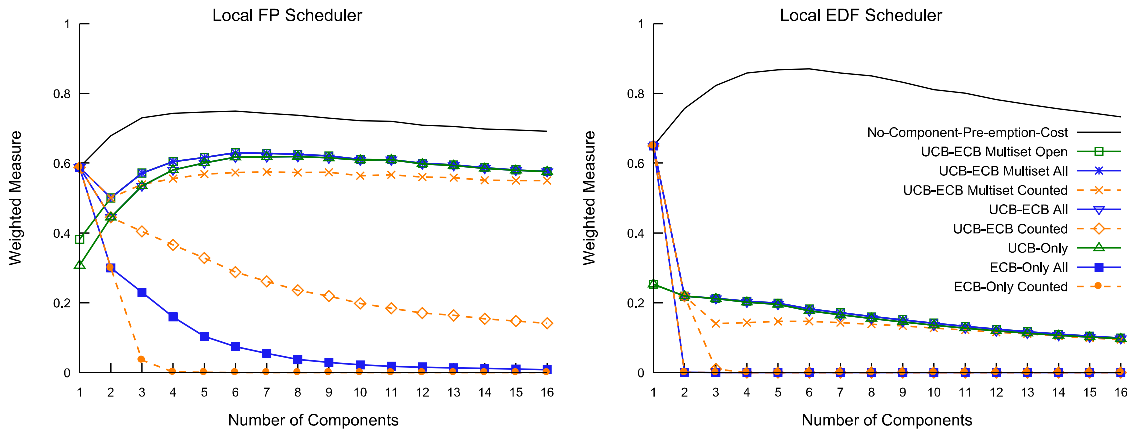


Fig. 9 Varying the number of components from 1 to 16, while keeping the number of tasks in the system fixed

Under a local EDF scheduler, all approaches that consider inter-component CRPD show a decrease in schedulability as the number of components increases above 2. The no-component-pre-emption-cost case shows an increase in schedulability up to approximately 6–7 components before decreasing. This is because as the number of components increases, the amount of intra-component CRPD from tasks in the same component decreases. This is then balanced against an increased delay in capacity from the components' servers. As the number of components is increased, and therefore the number of servers, Q^G is reduced leading to an increase in $P^G - Q^G$ which increases the maximum time between a server supplying capacity to its component. We also note that at two components, UCB-only, UCB–ECB-all and UCB–ECB-counted perform the same; as do the Multiset approaches. This is because the '-All' and '-Counted' variations are equivalent when there is only one other component.

9.2.2 System size

We investigated the effects of introducing components into a system by varying the system size from 1 to 10, see Fig. 10, where each increase introduces a new component which brings along with it 5 tasks taking up approximately twice the size of the cache.

When there is one component, all approaches except for UCB-only and UCB–ECB-multiset-open give the same result as no-component-pre-emption-cost. As expected, as more components are introduced into the system, system schedulability decreases for all approaches including no-component-pre-emption-cost. This is because each new component includes additional intra-component CRPD in addition to the inter-component CRPD that it causes when introduced. Furthermore, each new component that is introduced into the system effectively increases the maximum delay before search server supplies capacity to its components. Under a local FP scheduler, the ECB-only-all approach outperforms UCB–ECB-counted above a system size of 2, UCB-only and UCB–ECB-all outperform UCB–ECB-multiset-counted above a system size of 3, highlighting their incomparability. Again we note that the '-All' and '-Counted' variations are the same when there are only two components in the system.

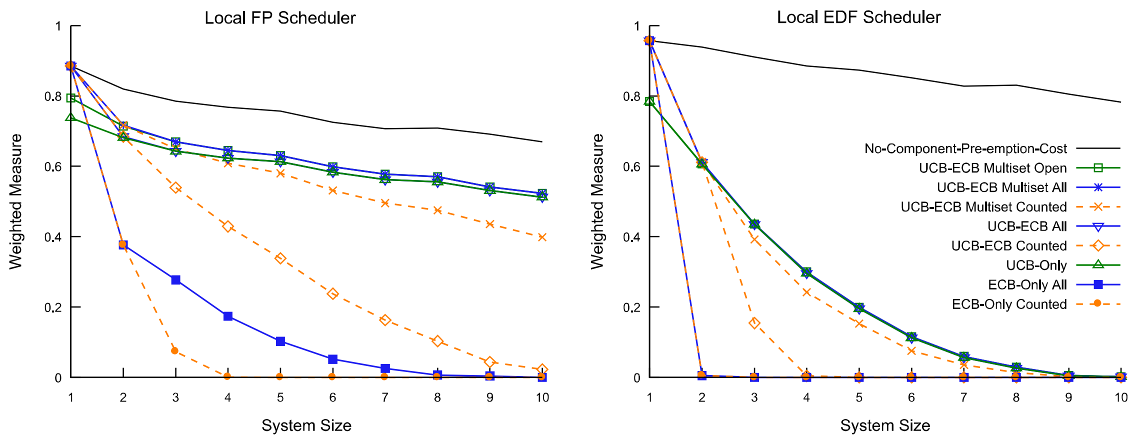


Fig. 10 Varying the system size from 1 to 10. An increase of 1 in the system size relates to introducing another component that brings along with it another 5 tasks and an increase in the cache utilisation of 2

9.2.3 Server period

The server period is a critical parameter when composing a hierarchical system. The results for varying the server period from 1 to 20 ms, with a fixed range of task periods from 10 to 1000 ms are shown in Fig. 11. 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 inter-component CRPD, up until approximately 7 ms under FP, and 7–8 ms under EDF, for the best performance. At this point although the inter-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. We note that in the case of EDF, the optimum server period is between 7 and 8 ms for most approaches and 9ms for the UCB–ECB-counted approach. This increase in optimum server period over FP is due to the increased calculated inter-component CRPD under a local EDF scheduler.

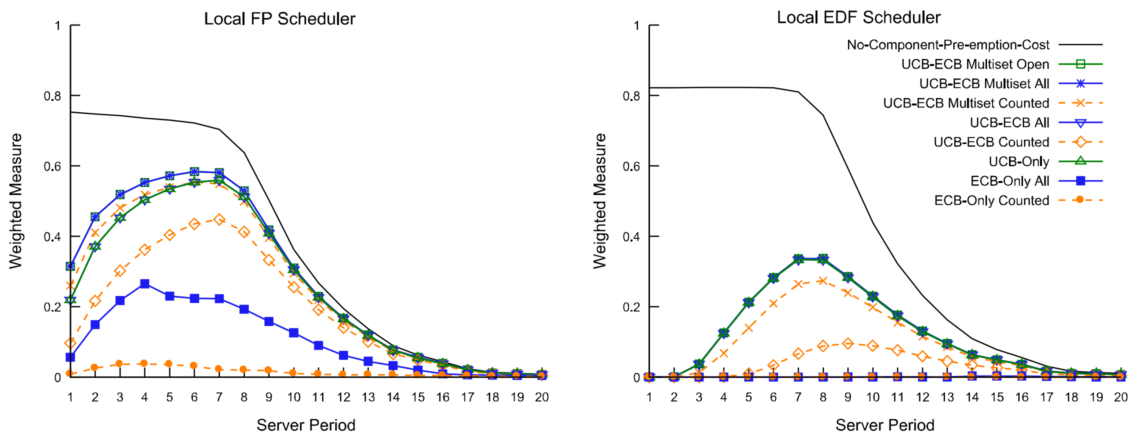


Fig. 11 Varying the server period from 1 to 20 ms (fixed task period range of 10–1000 ms)

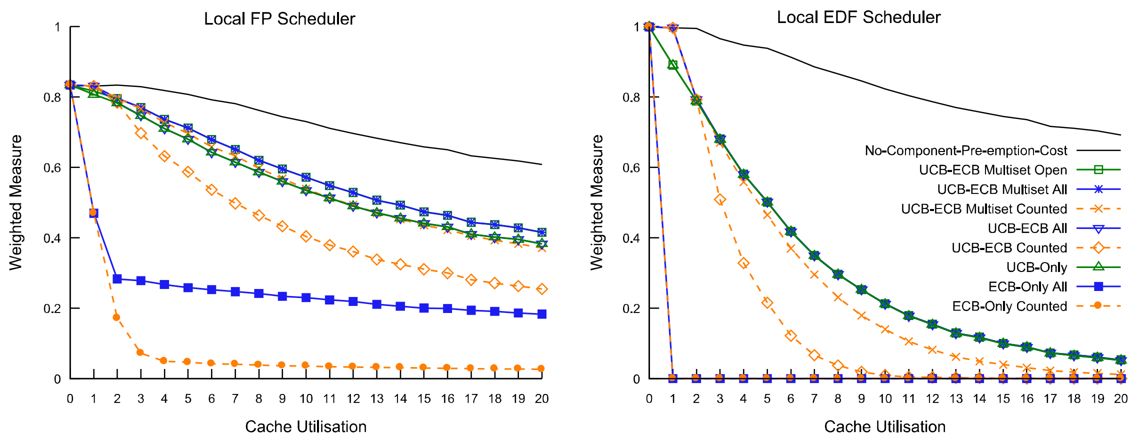


Fig. 12 Varying the cache utilisation from 0 to 20

9.2.4 Cache utilisation

As the cache utilisation increases the likelihood of the other components evicting UCBs belonging to the tasks in the suspended component increases. The results for varying the cache utilisation from 0 to 20 are shown in Fig. 12. In general, all approaches show a decrease in schedulability as the cache utilisation increases. Up to a cache utilisation of around 2, the UCB-Only and UCB–ECB-multiset-open approaches do not perform as well as the more sophisticated approaches, as the other components do not evict all cache blocks when they run. We also observe that up to a cache utilisation of 1 under a local FP scheduler, the ECB-only-counted, and the ECB-only-all approaches perform identically as no ECBs are duplicated.

We note that the weighted measure stays relatively constant for no-component-pre-emption-cost up to a cache utilisation of approximately 2.5. This is because the average cache utilisation of each component is still less than 1, which leads to relatively small intra-component CRPD between tasks.

9.2.5 Number of tasks

We also investigated the effect of varying the number of tasks, while keeping the number of components fixed. As we introduced more tasks, we scaled the cache utilisation in order to keep a constant ratio of tasks to cache utilisation. The results for varying the number of tasks from 3 to 48 are shown in Fig. 13. As expected, increasing the number of tasks leads to a decrease in schedulability across all approaches that consider inter-component CRPD. However, under a local EDF scheduler, the no-component-pre-emption-cost case actually shows an increase peaking at 12 tasks before decreasing due to the intra-component CRPD. Consider that when there are 3 tasks, there is only one task per component, so there is effectively no local scheduling. Therefore schedulability is based solely on the global scheduling algorithm, which is why the results for no-component-pre-emption-cost are the same for FP and EDF with 3 tasks. As more tasks are introduced the execution time of individual tasks is reduced, making it less likely that a task will miss a deadline due to its components' server not

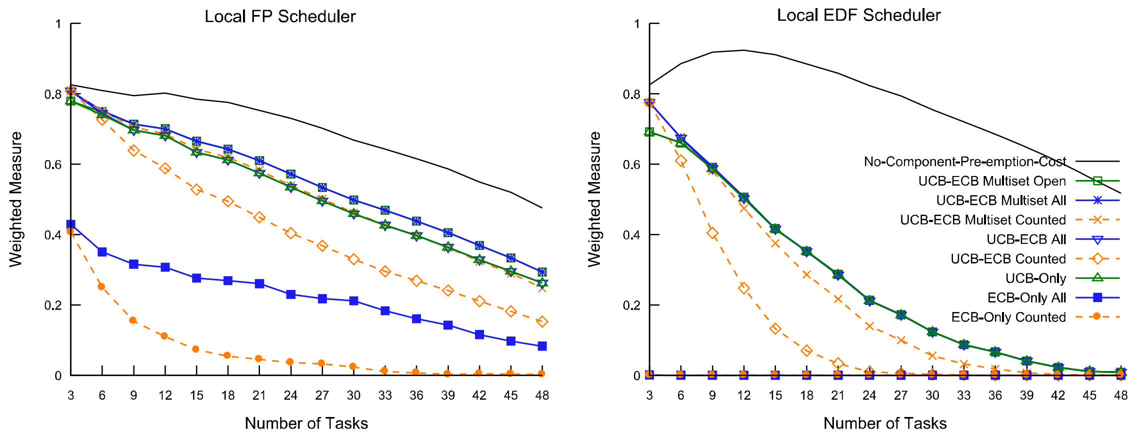


Fig. 13 Varying the total number of tasks from 3 to 48 (1–16 tasks per component)

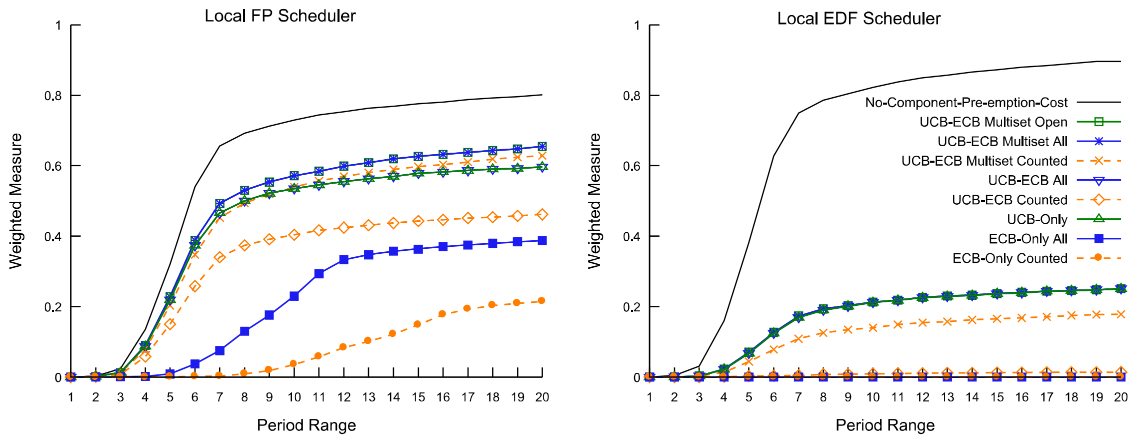


Fig. 14 Varying the period range of tasks from [1, 100] to [20, 2000]ms (while fixing the server period at 5 ms)

running. This increases schedulability until the effect of the intra-component CRPD outweighs it.

9.2.6 Task period range

We varied the range of task periods from [1, 100] to [20, 2000]ms, while fixing the server period at 5 ms. The results are shown in Fig. 14, as expected, the results show an increase in schedulability across all approaches as the task period range is increased.

9.3 EDF analysis investigation

The results for varying the system size, Fig. 10, and varying the cache utilisation, Fig. 12, suggest that the inter-component CRPD analysis for a local EDF scheduler has a significant reduction in performance when CRPD costs are increased. In this section we present the results for varying the BRT, which impacts the cost of a pre-emption, and for varying the deadlines of tasks. These results give further insight into this behaviour.

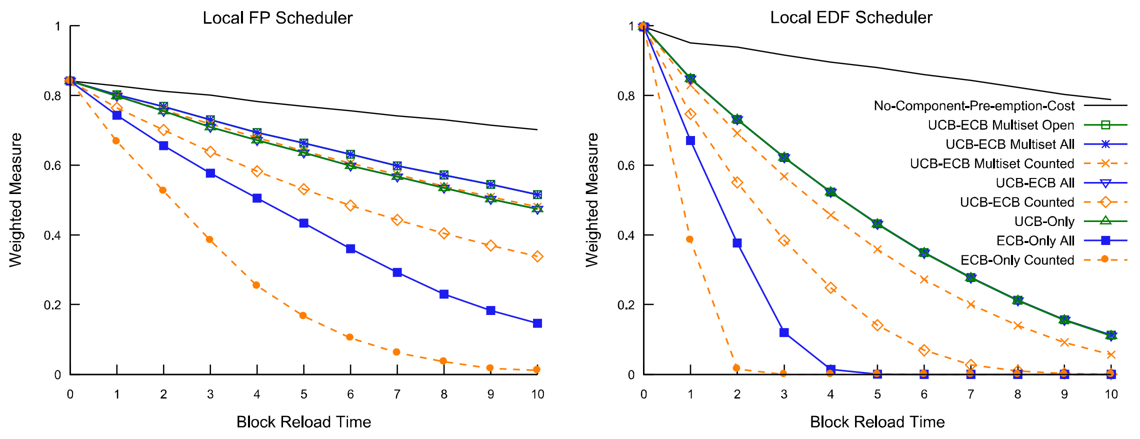


Fig. 15 Varying the block reload time (BRT) from 0 to 10 in steps of 1

9.3.1 Block reload time (BRT)

We investigated the effects of varying the BRT, effectively adjusting the costs of a pre-emption in Fig. 15. With a BRT of 0 there is effectively no CRPD, so all approaches achieve the same weighted measure. Once the BRT increases, the results show that the performance of the approaches that consider inter-component CRPD under a local EDF scheduler are significantly reduced. This indicates that the analysis for a local EDF scheduler is particularly susceptible to higher pre-emption costs.

9.3.2 Deadline factor

We also varied the task deadlines via $D_i = xT_i$ by varying x from 0.1 to 1 in steps of 0.1. The results are shown in Fig. 16. Under a local FP scheduler, all approaches showed an increase in the weighted measure as the deadlines are increased. Under a local EDF scheduler, the no-component-pre-emption-cost case performs as expected, showing an increase in schedulability as the deadlines are increased. Additionally, the non UCB–ECB-multiset approaches also show an increase in the number of schedulable systems. However, the UCB–ECB-multiset approaches show an increase in the number of systems deemed schedulable, and hence the weighted measure, up to a deadline

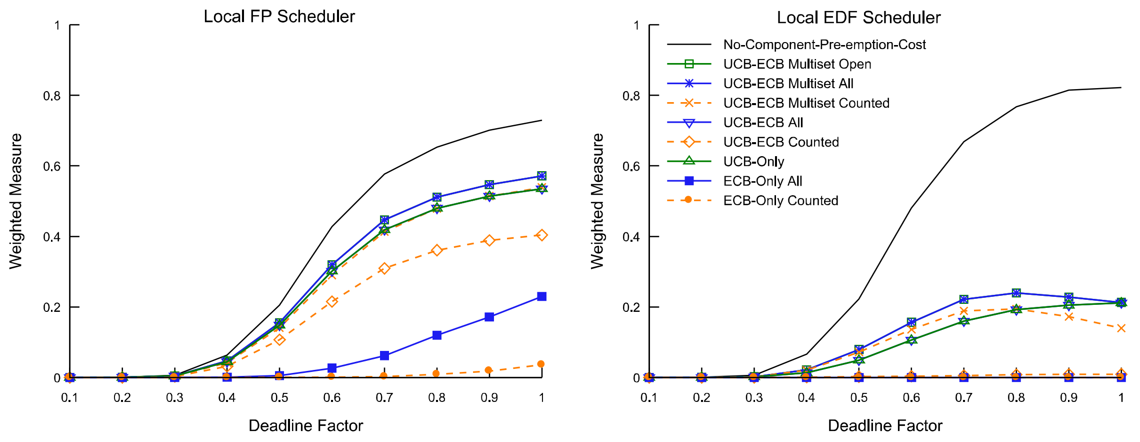


Fig. 16 Varying the task deadlines via $D_i = xT_i$ by varying x from 1 to 0.1 in steps of 0.1

factor of 0.8. After this point it shows a reduction in schedulability. This reduction is because although tasks deadlines are relaxed, and thus tasks are less likely to miss them, the number of times that the inter-component CRPD is accounted for is also increased as $E^G(D_k)E_k(t)$ will increase with longer deadlines.

9.4 Conclusion

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. The main contribution of this paper is a number of approaches for calculating cache related pre-emption delay (CRPD) in hierarchical systems with a global non-pre-emptive scheduler and a local pre-emptive FP or EDF scheduler. This is important because hierarchical scheduling has proved popular in industry as a way of composing applications from multiple vendors as well as re-using legacy code. However, unless the cache is partitioned, these isolated applications can interfere with each other, and so inter-component CRPD must be accounted for.

We presented a number of approaches to calculate inter-component CRPD in a hierarchical system with varying levels of sophistication. We 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 found the analysis for determining inter-component CRPD under a local EDF scheduler deemed a lower number of systems schedulable than the equivalent analysis for a local FP scheduler. This is due to pessimism in the analysis for EDF, and the difficulty in tightly bounding the number of server suspensions that result in inter-component CRPD. Specifically, the analysis considers the number of server suspensions that result in inter-component CRPD based on a task's deadline. In contrast for a local FP scheduler, the analysis can calculate a bound based on a task's response time.

While it was not the best approach in all cases we found the UCB–ECB-Multiset-Open approach, which does not require any information about the other components in the system, to be highly effective. This is a useful result as the approach does not require a closed system. Therefore it can be used when no knowledge of the other components is available and/or cache flushing is used between the execution of different components to ensure isolation and composability.

The UCB–ECB-multiset-all approach dominates the UCB–ECB-multiset-open approach. Therefore, if information about other components is available, it can be used to calculate tighter bounds in cases where not all cache blocks will be evicted by the other components. However, this requires a small enough cache utilisation such that the union of the other components ECBs is less than the size of the cache.

We note that the presented analysis is not dependent on the scheduling policies of the other components, and hence can be applied to a system where some components are scheduled locally by a FP scheduler while others use an EDF scheduler.

Previous works by Lipari and Bini (2005) and Davis and Burns (2008) have investigated how to select sever parameters. In the future, we intend to extend this work to find optimal server parameter settings taking into account inter-component CRPD. Lunniss et al. (2012) showed how the layout of tasks can be optimised to reduce CRPD. We also intend to extend this work to layout components and their tasks in order to reduce both intra- and inter-component CRPD so as to maximise system schedulability.

Acknowledgments 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, COST Action IC1202: Timing Analysis On Code-Level (TACLe) and the European Community's Seventh Framework Programme FP7 under Grant Agreement no. 246556, "RBUCE-UP". EPSRC Research Data Management: The benchmarks used were from external, publically available benchmark suites, no new primary data was created during this study.

References

Altmeyer S, Maiza C, Reineke J (2010) Resilience analysis: tightening the CRPD bound for set-associative caches. In: LCTES. New York, USA, pp 153–162

Altmeyer S, Davis RI, Maiza C (2011) Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. In: Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS). Vienna, Austria, pp 261–271

Altmeyer S, Davis RI, Maiza C (2012) Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. Real-Time Syst 48(5):499–512

ARINC (1991) ARINC 651: Design Guidance for Integrated Modular Avionics. Airlines Electronic Engineering Committee (AEEC)

ARINC (1996) ARINC 653: Avionics Application Software Standard Interface (Draft 15). Airlines Electronic Engineering Committee (AEEC)

Åsberg M, Behnam M, Nolte T (2013) An experimental evaluation of synchronization protocol mechanisms in the domain of hierarchical fixed-priority scheduling. In: Proceedings of the 21st International Conference on Real-Time and Network Systems (RTNS). Sophia Antipolis, France

Audsley NC, Burns A, Richardson M, Wellings AJ (1993) Applying new scheduling theory to static priority preemptive scheduling. Softw Eng J 8(5):284–292

Baruah SK, Mok AK, Rosier LE (1990a) Preemptive scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS). Lake Buena Vista, Florida, USA, pp 182–190

Baruah SK, Rosier LE, Howell RR (1990b) Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. Real-Time Syst 2(4):301–324

Bastoni A, Brandenburg B, Anderson J (2010) Cache-related preemption and migration delays: empirical approximation and impact on schedulability. In: Proceedings of Operating Systems Platforms for Embedded Real-Time applications (OSPERT). Brussels, Belgium, pp 33–44

Behnam M, Shin I, Nolte T, Nolin M (2007) SIRAP: a synchronization protocol for hierarchical resource sharing real-time open systems. In: Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT). pp 279–288

Bini E, Buttazzo G (2005) Measuring the performance of schedulability tests. Real-Time Syst 30(1):129–154

Burguière C, Reineke J, Altmeyer S (2009) Cache-related preemption delay computation for set-associative caches—pitfalls and solutions. In: Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET). Dublin, Ireland

Busquets-Mataix JV, Serrano JJ, Ors R, Gil P, Wellings A (1996) Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In: Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS). pp 204–212

Campoy AM, Sáez S, Perles A, Busquets JV (2004) Schedulability analysis in the EDF scheduler with cache memories. Lect Notes Comput Sci 2968:328–341

Davis RI, Zabos A, Burns A (2008) Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans Comput 57(9):1261–1276

Davis RI, Burns A (2008) An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In: Proceedings of the 16th International Conference on Real-Time and Network Systems (RTNS). Rennes, France, pp 19–28

Davis RI, Burns A (2005) Hierarchical fixed priority pre-emptive scheduling. In: Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)

Davis RI, Burns A (2006) Resource sharing in hierarchical fixed priority pre-emptive systems. In: Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS). Rio de Janeiro, Brazil, pp 257–270

Deng Z, Liu JWS (1997) Scheduling real-time applications in open environment. In: Proceedings of the IEEE Real-Time Systems Symposium (RTSS). San Francisco, USA

Dertouzos ML (1974) Control robotics: the procedural control of physical processes. In: Proceedings of the International Federation for Information Processing (IFIP) Congress. pp 807–813

Feng X, Mok AK (2002) A model of hierarchical real-time virtual resources. In: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS). Austin, TX, USA, pp 26–35

Fisher N, Dewan F (2012) A bandwidth allocation scheme for compositional real-time systems with periodic resources. Real-Time Syst 48(3):223–263

George L, Rivierre N, Spuri M (1996) Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report, INRIA

Ju L, Chakraborty S, Roychoudhury A (2007) Accounting for cache-related preemption delay in dynamic priority schedulability analysis. In: Design, Automation and Test in Europe Conference and Exposition (DATE). Nice, France, pp 1623–1628

Kuo T-W, Li C-H (1998) A fixed priority driven open environment for real-time applications. In: Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS). Madrid, Spain

Lee C, Hahn J, Seo Y, Min S, Ha H, Hong S, Park C, Lee M, Kim C (1998) Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. IEEE Trans Comput 47(6):700–713

Leung JY-T, Merrill ML (1980) A note on preemptive scheduling of periodic, real-time tasks. Inf Process Lett 11(3):115–118

Lipari G, Bini E (2005) A methodology for designing hierarchical scheduling systems. J Embed Comput 1(2):257–269

Lipari G, Baruah SK (2000a) Efficient scheduling of real-time multi-task applications in dynamic systems. In: Proceddings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS). pp 166–175

Lipari G, Carpenter J, Baruah S (2000b) A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments. In: Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS). Orlando, FL, USA, pp 217–226

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61

Lunniss W, Altmeyer S, Davis RI (2012) Optimising task layout to increase schedulability via reduced cache related pre-emption delays. In: Proceedings of the International Conference on Real-Time Networks and Systems (RTNS). Pont à Mousson, France, pp 161–170

Lunniss W, Altmeyer S, Maiza C, Davis RI (2013) Intergrating cache related pre-emption delay analysis into edf scheduling. In: Proceedings 19th IEEE Converence on Real-Time and Embedded Technology and Applications (RTAS). Philadelphia, USA, pp 75–84

Lunniss W, Altmeyer S, Lipari G, Davis RI (2014a) Accounting for cache related pre-emption delays in hierarchical scheduling. In: Proceedings of the 22nd International Conference on Real-Time Networks and Systems (RTNS). Versailles, France, pp 183–192

Lunniss W, Altmeyer S, Davis RI (2014b) 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). Versailles, France

Richter K (2005) Compositional scheduling analysis using standard event models. PhD Dissertation, Technical University Carolo-Wilhelmina of Braunschweig

Ripoll I, Crespo A, Mok AK (1996) Improvement in feasibility testing for real-time tasks. Real-Time Syst 11(1):19–39

Saewong S, Rajkumar R, Lehoczky J, Klein M (2002) Analysis of hierarchical fixed priority scheduling. In: Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS). Vienna, Austria, pp 173–181

Shin I, Lee I (2003) Periodic resource model for compositional real-time guarantees. In: Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS). Cancun, Mexico, pp 2–13

Spuri M (1996) Analysis of deadline schedule real-time systems. Technical Report, INRIA

Staschulat J, Schliecker S, Ernst R (2005) Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In: Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS). Balearic Islands, Spain, pp 41–48

Tan Y, Mooney V (2007) Timing analysis for preemptive multitasking real-time systems with caches. ACM Trans Embed Comput Syst 6(1):7

Watkins CB, Walter R (2007) Transitioning from federated avionics architectures to integrated modular avionics. In: Proceedings of the 26th IEE/AIAA Digital Avionics Systems Conference (DASC)

Xu M, Phan LTX, Lee I, Sokolsky O, Xi S, Lu C, Gill C (2013) Cache-aware compositional analysis of real-time multicore virtualization platforms. In: Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS). Vancouver, Canada

Zhang F, Burns A (2009) Schedulability analysis for real-time systems with EDF scheduling. IEEE Trans Comput 58(9):1250–1258

Will Lunniss was awarded an EngD in Computer Science from the University of York in 2014. His thesis focused on the effects of cache related pre-emption delays in embedded real-time systems. Will is currently a Software Engineer at Rapita Systems Ltd. A spin out company from the University of York, specialising in the verification of embedded real-time systems for the aerospace and automotive industries.

Sebastian Altmeyer is a post-doctoral researcher in the LASSY Group of the University of Luxembourg since 2015 and was part of the Computer Systems Architecture Group at the University of Amsterdam, Netherlands from 2013 to 2015. He received his PhD in Computer Science in 2012 from Saarland University in Saarbruecken, Germany with a thesis on the analysis of preemptively scheduled hard real-time systems. His research interests are the analysis and verification of hard realtime systems in general, with a particular focus on worst-case timing analysis and real-time scheduling.

Giuseppe Lipari graduated in Computer Engineering from the University of Pisa (Italy) in 1996. He received his PhD in Computer Engineering from the Scuola Superiore Sant'Anna of Pisa in 2000, where he also worked as Associate Professor until 2014. From 2012, he spent two years at the École Normale Supérieure de Cachan (France) as a Marie-Curie fellow. From 2014, he is Full Professor of Computer Science at the University of Lille, where he is part of the Embedded Real-Time Adaptative system Design and Execution (Émeraude) team of the Centre de Recherche en Informatique, Signal et Automatique (CRIStAL) of Lille, and of the Institut de Recherche en Composants logiciels et matériels pour l'Information et la Communication Avancée (IRCICA). His research interests span over multiple domains related to embedded systems: real-time operating systems, scheduling algorithms, formal methods, timed automata, and wireless sensor networks. During his career, he has co-authored about one hundred papers in peer-reviewed scientific journals and proceedings of international conferences. He is associate Editor of the Real-Time Systems Journal and of the IEEE Transactions on Computers.

Robert I. Davis is a Senior Research Fellow in the Real-Time Systems Research Group at the University of York, UK. Robert also holds an International Chair with the AOSTE team at Inria Paris-Rocquencourt, France. He received his DPhil in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial products. Robert's research interests include scheduling algorithms and schedulability analysis for real-time systems and networks.