Accounting for Cache Related Pre-emption Delays in Hierarchical Scheduling

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

[1]Department of Computer Science, University of York, York, UK
{wl510,rob.davis}@york.ac.uk

[2]Computer Systems Architecture Group, University of Amsterdam, Netherlands
altmeyer@uva.nl

[3]Scuola Superiore Sant'Anna, IT
g.lipari@sssup.it

[4]LSV – ENS Cachan, FR
giuseppe.lipari@lsv.ens-cachan.fr

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. While one solution is to flush the cache after every application context switch, this can potentially lead to a degradation in performance. 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.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems - *Real-time and embedded systems*

Keywords

Cache Related Pre-emption Delays, Hierarchical scheduling, Fixed priority pre-emptive scheduling, Response time analysis

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 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.

RTNS '14, October 08 - 10 2014, Versailles, France
http://dx.doi.org/10.1145/2659787.2659797

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. It is therefore important to be able to calculate, and account for, CRPD when determining if a system is schedulable or not. This is further complicated when using hierarchical scheduling as servers will often be suspended while their components' tasks are still active, that is they have started but have not yet completed execution. While a server is suspended, the cache can be polluted by the tasks belonging to other components. When the global scheduler then switches back to the first server, tasks belonging to the associated component may have to reload blocks into cache that were in use before the global context switch.

Hierarchical scheduling has been studied extensively in the past 15 years. Deng and Liu [15] were the first to propose such a two-level scheduling approach. Later Feng and Mok [16] proposed the resource partition model and schedulability analysis based on the supply bound function. Shin and Lee [25] introduced the concept of a temporal interface and the periodic resource model, and refined the analysis of Feng and Mok. Kuo and Li [17] and Saewong *et al*. [24] specifically focused on fixed priority hierarchical scheduling. Lipari and Bini [19] solved the problem of computing the values of the partition parameters to make an application schedulable. Davis and Burns [12] proposed a method to compute the response time of tasks running on a local fixed priority scheduler. Later, Davis and Burns [11] investigated selecting optimal server parameters for fixed priority pre-emptive hierarchical systems.

Hierarchical systems have been used mainly in the avionics industry. The IMA (Integrated Modular Avionics) [28], [3] 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 [4]. The ARINC 653 standard [4] defines temporal partitioning for avionics applications. The global scheduler is a simple Time Division Multiplexing (TDM), in which time is divided into frames of fixed length, each frame is divided into slots and each slot is assigned to one application.

Analysis of CRPD uses the concept of *useful cache blocks* (UCBs) and *evicting cache blocks* (ECBs) based on the work by Lee *et al.* [18]. Any memory block that is accessed by a task while executing is classified as an ECB, as accessing that block may evict a cache block of a pre-empted task. Out of the set of ECBs, some of them may also be UCBs. A memory block m is classified as a UCB at program point ρ, if (i) m may be cached at ρ and (ii)

m may be reused at program point q that may be reached from ρ without eviction of m on this path. In the case of a pre-emption at program point ρ, only the memory blocks that are (i) in cache and (ii) will be reused, may cause additional reloads. For a more thorough explanation of UCBs and ECBs, see section 2.1 "Pre-emption costs" of [2].

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 fixed priority pre-emptive scheduling under a flat, single-level system. They include Lee *et al.* [18] UCB-Only approach, which considers just the pre-empted task(s), and Busquets *et al.* [10] 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 [27] UCB-Union approach, Altmeyer *et al.* [1] ECB-Union approach, and an alternative approach by Staschulat *et al.* [26]. Finally, there are advanced multiset based approaches that consider the pre-empted and pre-empting task(s) by Altmeyer *et al.* [2], ECB-Union Multiset, UCB-Union Multiset, and a combined multiset approach. This analysis has also been recently been adapted to pre-emptive EDF scheduling by Lunniss *et al.* [22].

Xu *et al.* [29] 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.

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 flat single-level systems, and schedulability analysis for hierarchical systems. Section 4 introduces the new analysis for calculating CRPD due to hierarchical scheduling. Section 5 evaluates the analysis using case study data, and section 6 evaluates it using synthetically generated tasksets. Finally, section 7 concludes with a summary and outline of future work.

2. SYSTEM MODEL

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

We assume a single processor system comprising m applications or components, each with a dedicated server ($S^1..S^m$) that allocates processor capacity to it. We use Ψ to represent the set of all components in the system. G is used to indicate the index of the component that is being analysed. Each server S^G has a budget Q^G and a period P^G, such that the associated component will receive Q^G units of execution time from its server every P^G units of time. Servers are assumed to be scheduled globally using a non-pre-emptive scheduler, as found in systems that use time partitioning to divide up access to the processor. While a server has remaining capacity and is allocated the processor, we assume that the tasks of the associated component are scheduled pre-emptively according to their fixed priorities. If there are no tasks to schedule, we assume that the processor idles until the server exhausts all of its capacity, or a new task is released.

The system comprises a taskset Γ made up of a fixed number of tasks ($\tau_1..\tau_n$) divided between the components. 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 [13] [8] [5]. Resource sharing and its effects on CRPD have also been studied for single level systems [1] [2]. 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. We assume that deadlines are *constrained* (i.e. $D_i \leq T_i$). We used 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.

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 $\text{ECB}_1 = \{1,2,3,4\}$ and $\text{UCB}_1 = \{2,4\}$. Each component G also has a set of UCBs, UCB^G and a set of ECBs, ECB^G, that contain respectively all of the UCBs, and all of the ECBs, of their tasks, i.e. $\text{UCB}^G = \bigcup_{\forall \tau_i \in \Gamma^G} \text{UCB}_i$ and $\text{ECB}^G = \bigcup_{\forall \tau_i \in \Gamma^G} \text{ECB}_i$.

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

We focus on instruction only caches. In the case of data caches, the analysis would either require a write-through cache or further extension in order to be applied to write-back caches. We also assume that tasks do not share any code.

We assume a direct mapped cache, but the analysis can be extended to set-associative LRU caches as described in section II. A "*Set-associative Caches"* of the technical report on which this paper is based [21].

3. EXISTING SCHEDULABILITY AND CRPD ANALYSIS

In this section we briefly recap how CRPD can be calculated in a flat, single-level system, and how schedulability analysis without CRPD analysis can be performed for hierarchical systems. 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 [6]:

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

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 [14].

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.* [2]. In this work, we use the Combined Multiset approach by Altmeyer *et al.* [2] 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:

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

3.1 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.

3.1.1 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* [25], 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, i.e. 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,* for component *G* as $isbf^G$ [23]:

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

Integrating (3) into equation (1) gives the response time of τ_i under server S^G taking into account the shared access to the processor as:

$$R_i^{\alpha+1} = isbf^G\left(C_i + \sum_{\forall j \in hp(G,i)} \left\lceil \frac{R_i^{\alpha} + J_j}{T_j} \right\rceil C_j \right) \quad (4)$$

4. CRPD ANALYSIS FOR HIERARCHICAL SYSTEMS

In this section, we describe how CRPD analysis can be extended for use in hierarchical systems 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 [10], [18], [27] and [2] respectively to hierarchical systems. This analysis assumes a non-pre-emptive global scheduler (i.e. the capacity of a server is supplied without pre-emption, but may be supplied starting at any time during the server's period), and a pre-emptive fixed priority local scheduler.

We will explain a number of different methods, building up in complexity.

The analysis needs to capture the cost of reloading any UCBs into cache that were evicted by tasks belonging to other components. This can be achieved by combining the intra-component CRPD due to pre-emptions between tasks within the same component, (2), with the modified response time analysis for non-dedicated

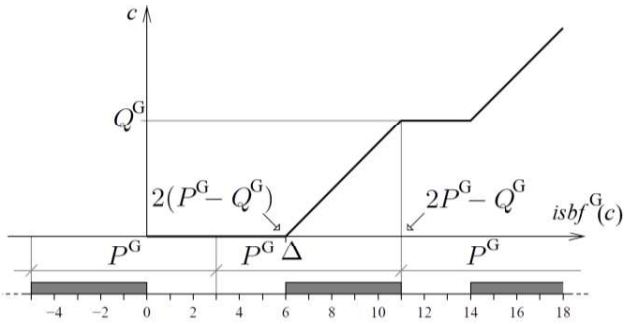

Figure 1. General case of a server where $Q^G = 5$ and $P^G = 8$ showing it can take up to 6 time units before a task receives supply

processor access, (4), with a new term, γ_i^G :

$$R_i^{\alpha+1} = isbf^G\left(C_i + \sum_{\forall j \in hp(G,i)} \left(\left\lceil \frac{R_i^{\alpha} + J_j}{T_j} \right\rceil C_j + \gamma_{i,j} \right) + \gamma_i^G \right) \quad (5)$$

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.

We use $E^G(R_i)$ to denote the maximum number of times server S^G can be both suspended and resumed during the response time of task τ_i:

$$E^G(R_i) = 1 + \left\lfloor \frac{R_i}{P^G} \right\rfloor \quad (6)$$

We use the term *disruptive execution* to describe an execution of server S^Z while server S^G is suspended that results in tasks from component Z evicting cache blocks that tasks in component G might have loaded and 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 and as such, the number of disruptive executions is bounded by the number of times that server S^G can be both suspended and resumed. Specifically, we are interested in how many disruptive executions a server can have that impact a particular task τ_i. We use X^Z to denote the maximum number of such disruptive executions.

$$X^Z(S^G, R_i) = \min\left(E^G(R_i), 1 + \left\lceil \frac{R_i}{P^Z} \right\rceil \right) \quad (7)$$

4.1 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.

4.1.1 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. If *Z* is a specific component, then we can calculate the CRPD impacting task τ_i of component *G* due to the other components in the system as:

$$\gamma_i^G = \text{BRT} \bullet E^G(R_i) \bullet \left| \bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \text{ECB}^Z \right| \quad (8)$$

4.1.2 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 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, $X^Z\left(S^G, R_i\right)$. 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:

$$\gamma_i^G = \text{BRT} \bullet \sum_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(X^Z\left(S^G, R_i\right) \bullet \left| \text{ECB}^Z \right| \right) \quad (9)$$

Note that the ECB-Only-All and ECB-Only-Counted approaches are incomparable.

4.2 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 the maximum number of UCBs over all tasks in component G that may pre-empt task τ_i, and task τ_i itself, i.e. $\tau_k \in \text{hep}(G,i)$.

$$\bigcup_{\forall k \in \text{hep}(G,i)} \text{UCB}_k \quad (10)$$

We then multiply the number of UCBs (10) by the number of times that server S^G can be both suspended and resumed during the response time of task τ_i.

$$\gamma_i^G = \text{BRT} \bullet E^G(R_i) \bullet \left| \bigcup_{\forall k \in \text{hep}(G,i)} \text{UCB}_k \right| \quad (11)$$

This approach is incomparable with the ECB-Only-All and ECB-Only-Counted approaches.

4.3 UCB-ECB

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

4.3.1 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 can calculate which UCBs could be evicted by the tasks in the other components, (10). 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.

$$\left| \left(\bigcup_{\forall k \in \text{hep}(G,i)} \text{UCB}_k \right) \cap \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \text{ECB}^Z \right) \right| \quad (12)$$

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:

$$\gamma_i^G = \text{BRT} \bullet E^G(R_i) \bullet \left| \left(\bigcup_{\forall k \in \text{hep}(G,i)} \text{UCB}_k \right) \cap \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \text{ECB}^Z \right) \right| \quad (13)$$

By construction, the UCB-ECB-All approach dominates the ECB-Only-All and UCB-Only approaches.

4.3.2 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:

$$\left| \left(\bigcup_{\forall k \in \text{hep}(G,i)} \text{UCB}_k \right) \cap \text{ECB}^Z \right| \quad (14)$$

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:

$$\gamma_i^G = \text{BRT} \bullet \sum_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(X^Z\left(S^G, R_i\right) \bullet \left| \left(\bigcup_{\forall k \in \text{hep}(G,i)} \text{UCB}_k \right) \cap \text{ECB}^Z \right| \right) \quad (15)$$

By construction, the UCB-ECB-Counted approach dominates the ECB-Only-Counted approach, but is incomparable with the UCB-Only approach.

4.4 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 $\tau_k \in \text{hep}(G,i)$ 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 $\tau_k \in \text{hep}(G,i)$ can run during the response time of τ_i and then multiplying 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, i.e. $E^G(R_k)$.

4.4.1 UCB-ECB-Multiset-All

First we form a multiset that contains the UCBs of task τ_k repeated $E^G(R_k)E_k(R_i)$ times for each task $\tau_k \in \text{hep}(G,i)$.

$$M_{G,i}^{ucb} = \bigcup_{\forall k \in \text{hep}(G,i)} \left(\bigcup_{E^G(R_k)E_k(R_i)} \text{UCB}_k \right) \quad (16)$$

Then we form a second multiset 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.

$$M_{G,i}^{ecb-A} = \bigcup_{E^G(R_i)} \left(\bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \text{ECB}^Z \right) \quad (17)$$

The total CRPD incurred by task τ_i, in component G due to the other components in the system is then given by the size of the multiset intersection of $M_{G,i}^{ucb}$ (16) and $M_{G,i}^{ecb-A}$ (17).

$$\gamma_i^G = \text{BRT} \bullet \left| M_{G,i}^{ucb} \cap M_{G,i}^{ecb-A} \right| \quad (18)$$

4.4.2 UCB-ECB-Multiset-Counted

For the UCB-ECB-Multiset-Counted approach, we keep equation (16) for calculating the set of UCBs; however, we form a second multiset that contains $X^Z\left(S^G, R_i\right)$ 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 $X^Z\left(S^G, R_i\right)$ times within the response time of task τ_i.

$$M_{G,i}^{ecb-C} = \bigcup_{\substack{\forall Z \in \Psi \\ \wedge Z \neq G}} \left(\bigcup_{X^Z\left(S^G, R_i\right)} \text{ECB}^Z \right) \quad (19)$$

The total CRPD incurred by task τ_i, in component G due to the other components in the system is then given by the size of the multiset intersection of $M_{G,i}^{ucb}$ (16) and $M_{G,i}^{ecb-C}$ (19).

$$\gamma_i^G = \text{BRT} \bullet \left| M_{G,i}^{ucb} \cap M_{G,i}^{ecb-C} \right| \quad (20)$$

4.4.3 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, and as such, the precise number of ECBs does not matter.

For the UCB-ECB-Multiset-Open approach, we keep equation (16) for calculating the set of UCBs. Further, we form a second multiset 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.

$$M_{G,i}^{ecb-O} = \bigcup_{E^G(R_i)} \left(\{1,2,..N\} \right) \quad (21)$$

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 given by the size of the multiset intersection of $M_{G,i}^{ucb}$ (16) and $M_{G,i}^{ecb-O}$ (21).

$$\gamma_i^G = \text{BRT} \bullet \left| M_{G,i}^{ucb} \cap M_{G,i}^{ecb-O} \right| \quad (22)$$

4.5 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. Figure 2 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. The diagram also highlights dominance. For example, UCB-ECB-Multiset-All dominates UCB-ECB-Multiset-Open and UCB-ECB-All, and UCB-All dominates ECB-Only-All.

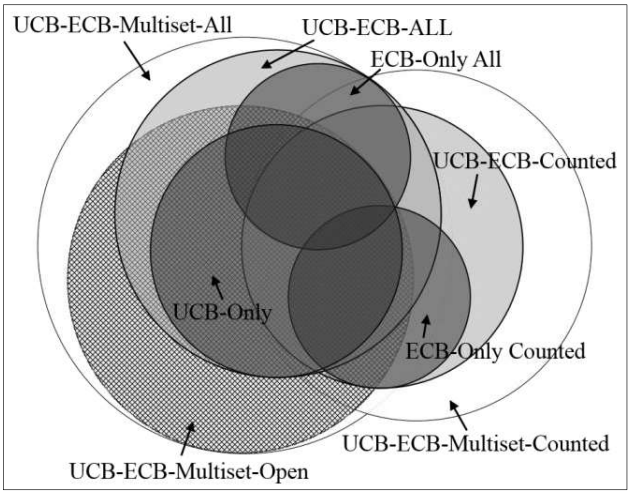

Figure 2. Venn diagram showing the relationship between the different approaches.

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, $E^G(R_1)=10$, $X^A\left(S^G, R_1\right)=10$, $X^B\left(S^G, R_1\right)=2$. $ECB^A=\{1,2\}$ 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 assuming that component B runs at the same rate as component G:

$$\begin{aligned} \gamma_1^G &= 1 \times 10 \times \left| \{1,2\} \cup \{3,4,5,6,7,8,9,10\} \right| \\ &= 10 \times \left| \{1,2,3,4,5,6,7,8,9,10\} \right| = 10 \times 10 = 100 \end{aligned}$$

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.

$$\begin{aligned} \gamma_1^G &= 1 \times \begin{pmatrix} 10 \times \left| \{1,2\} \right| + \\ 2 \times \left| \{3,4,5,6,7,8,9,10\} \right| \end{pmatrix} \\ &= (10 \times 2) + (2 \times 8) = 36 \end{aligned}$$

Below, we present a more detailed worked example for all approaches where the ECB-Only-All approach outperforms the ECB-Only-Counted approach, which confirms the incomparability of the –All and –Counted approaches.

Figure 3 shows an example schedule for four components, G, A, B and C, where component G has two tasks. Let BRT=1, $E^G(R_1)=1$, $E^G(R_2)=2$ and $E_1(R_2)=1$ and the number of disruptive executions be:

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

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

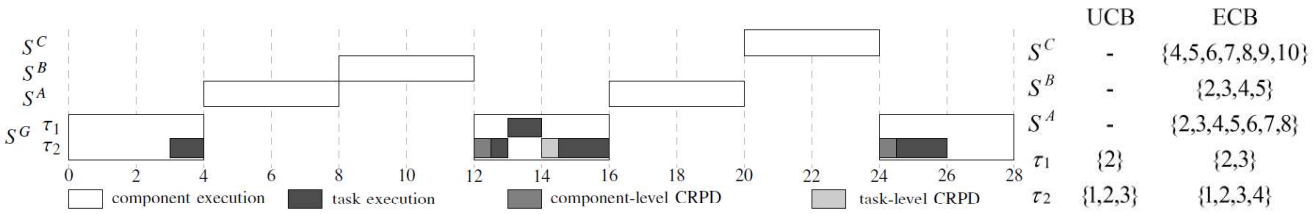

Figure 3. Example schedule and UCB/ECB data to demonstrate how the different approaches work.

ECB-Only-All calculates:

$$\gamma_2^G = 1 \times 2 \times \left|\{2,3,4,5,6,7,8\} \cup \{2,3,4,5\} \cup \{4,5,6,7,8,9,10\}\right| = 2 \times \left|\{2,3,4,5,6,7,8,9,10\}\right| = 2 \times 9 = 18$$

ECB-Only-Counted:

$$\gamma_2^G = 1 \times \begin{pmatrix} 2 \times \left|\{2,3,4,5,6,7,8\}\right| + \\ 2 \times \left|\{2,3,4,5\}\right| + \\ 2 \times \left|\{4,5,6,7,8,9,10\}\right| \end{pmatrix} = (2 \times 7) + (2 \times 4) + (2 \times 7) = 36$$

UCB-Only:

$$\gamma_2^G = 1 \times 2 \times \left(\left|\{2\} \cup \{1,2,3\}\right|\right) = 2 \times \left|\{1,2,3\}\right| = 6$$

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 by using the more sophisticated approaches, for example, UCB-ECB-Multiset-Counted:

$$M_{G,2}^{ucb} = \{2\} \cup \{1,2,3\} \cup \{1,2,3\} = \{1,1,2,2,2,3,3\}$$

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

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

For the tightest bound in this specific case, the UCB-ECB-Multiset-All approach does the best:

$$M_{G,2}^{ecb-A} = \bigcup_2 \left(\{2,3,4,5,6,7,8\} \cup \{2,3,4,5\} \cup \{4,5,6,7,8,9,10\}\right) = \bigcup_2 \left(\{2,3,4,5,6,7,8,9,10\}\right) = \{2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10\}$$

$$\gamma_2^G = 1 \times \left|M_{G,2}^{ucb} \cap M_{G,2}^{ecb-A}\right| = 1 \times \left|\{2,2,3,3\}\right| = 4$$

Assuming there are 12 cache sets in total[1], the UCB-ECB-Multiset-Open approach gives:

$$M_{G,2}^{ecb-O} = \bigcup_2 \left(\{1,2,3,4,5,6,7,8,9,10,11,12\}\right) = \{1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,11,11,12,12\}$$

$$\gamma_2^G = 1 \times \left|M_{G,2}^{ucb} \cap M_{G,2}^{ecb-O}\right| = 1 \times \left|\{1,1,2,2,3,3\}\right| = 6$$

[1] 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.

5. 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[2] 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[3] 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 equation (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.

We assigned a deadline of 2 ms to all of the interrupt tasks, and implicit deadlines i.e. $D_i = T_i$, 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 arrangement of tasks in each component, using the following technique. We split the normal tasks at random into 3 components with four tasks in two components and five in the other, and then assigned task priorities according to Deadline Monotonic priority assignment. Next we set the period of each component's server to 12.5ms (half the minimum task period). Finally we organised tasks in each component in memory in a sequential order based on their

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

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

priority, and then ordered components in memory sequentially based on their index.

5.1 Success Ratio

For each system, the total task utilization across all tasks not including pre-emption cost was varied from 0.025 to 1 in steps of 0.025. For each utilization value, we initialised each servers' capacity to the minimum possible value, (i.e. the utilisation of all of its tasks). We then performed a binary search between this minimum and the maximum, (i.e. 1 minus the minimum utilisation of all of the other components) until we found the server capacity required to make the component schedulable. As the servers all had equal periods, provided all components were schedulable and the total capacity required by all servers was ≤ 100%, then the system was deemed schedulable at that specific utilisation level. In addition to evaluating each of the presented approaches, 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 the combined multiset approach as it is the most effective approach available [2].

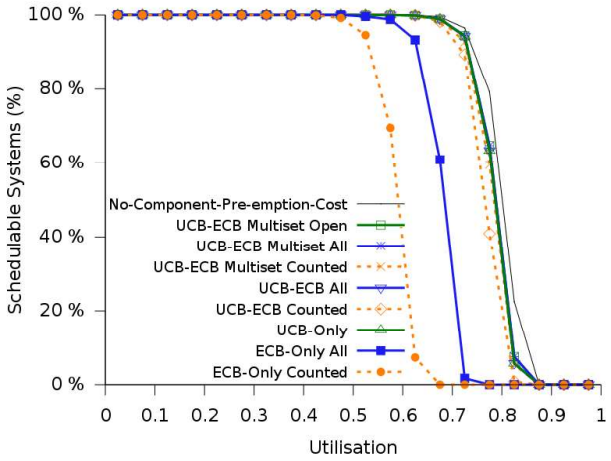

Figure 4. Percentage of schedulable tasksets at each utilisation level for the case study tasksets

The results for the case study are shown in Figure 4, note that the graphs in this paper 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. 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. Next was UCB-ECB-Counted which though it considers all components, accounts for the other components pessimistically in most cases. The remainder of the approaches all performed very similarly. We note that No-Component-Pre-emption-Cost reveals that the component pre-emption cost is very small for the PapaBench tasks, 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.

6. EVALUATION

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

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

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

To generate the components and tasksets, we generated n (default of 24) tasks using the UUnifast algorithm [9] 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 $C_i = U_i\ T_i$, and implicit deadlines were set, i.e. $D_i = T_i$. We used the UUnifast algorithm to obtain the number of ECBs for each task so that the ECBs added up to the desired cache utilisation (default of 10). Here, cache utilisation describes the ratio of the total size of the tasks to the size of the cache. A cache utilisation of 1 means that the tasks fit exactly in the cache, whereas a cache utilisation of 10 means the total size of the tasks is 10 times the size of the cache. The number of UCBs was chosen at random between 0 and 30% of the number of ECBs on a per task basis, and the UCBs were placed in a single group at a random location in each task.

We then split the tasks at random into 3 components with equal numbers of tasks in each, and assigned task priorities according to Deadline Monotonic priority assignment. Next we set the period of each component's server to 5ms. Finally we organised tasks in each component in memory in a sequential order based on their priority, and then ordered components in memory sequentially based on their index. We generated 1000 systems using this technique. It took approximately 5-10 seconds to analyse a single taskset under all approaches over the range of utilisation levels for the base line configuration using a single core of a 2.8GHz AMD Opteron 6386 SE.

6.1 Success Ratio

We determined the schedulability of the synthetic tasksets using the same approach described in the first paragraph of section 5.1.

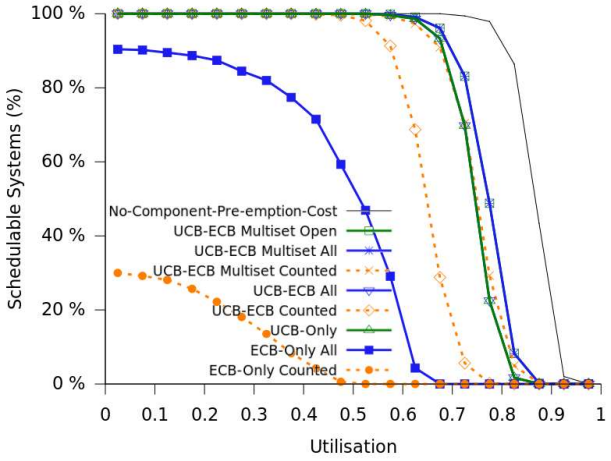

Figure 5. Percentage of schedulable tasksets at each utilisation level for the synthetic tasksets

The results for the baseline evaluation are shown in Figure 5. The ECB-Only-Counted approach is least effective as it only considers the other components and does so individually. ECB-Only-All was next, followed by UCB-ECB-Counted. 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. Finally, the UCB-ECB-Multiset-All and UCB-ECB-Multiset-Open approaches performed the best. Despite only considering the properties of the component under analysis, the UCB-ECB-Multiset-Open approach also 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.

6.2 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 [7] 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.

To investigate the effect of key cache and taskset configurations we varied the following 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)

We used 100 systems for each utilisation level from 0.025 to 1.0 in steps of 0.025 for the weighted schedulability experiments. The results and explanations for varying the cache utilisation, number of tasks, and range of task periods are available in the appendix of the technical report [21] on which this paper is based.

6.2.1 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 (24 tasks in one component) to 24 (1 task per component), see Figure 6. 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 component CRPD because they assume that every time the single component is suspended, its UCBs are evicted, even though there is only one component running in the system. At 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 get rapidly worse as they over-count blocks. All other approaches improve as the number of components is increased above 2 up to 8 components. This is because as the number of components increases, the amount of intra-component CRPD from tasks in the same component decreases. While the higher number of components does lead to increased inter-component CRPD, due to higher number of server context switches, it is not enough to cancel out the gains from reduced intra-component CRPD. This is because the increasing number of components, which are scheduled non-pre-emptively, is reducing the overall amount of pre-emption in the system. However, above 8 components, schedulability decreases as the inter-component CRPD and server delays become the dominant factors. 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.

6.2.2 System Size

We investigated the effects of introducing components into a system by varying the system size from 1 to 10, see Figure 7, 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 component includes additional intra-component CRPD, in addition to the inter-component CRPD that it causes when introduced. Notably, 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 4, highlighting their incomparability. Again we note that the '-All' and '-Counted' variations are the same when there are only two components in the system.

6.2.3 Server Period

The server period is a critical parameter when composing a hierarchical system. The results for varying the server period from 1ms to 20ms, with a fixed range of task periods from 10 to 1000ms are shown in Figure 8. When the component pre-emption costs are ignored, having a small server period ensures that short deadline tasks meet their time constraints. However, switching between components clearly has a cost associated with it making it desirable to switch as infrequently as possible. As the server period increases, schedulability increases due to a smaller number of server context switches, and hence component CRPD, up until

around 7ms for the best performance. At this point, although the component CRPD continues to decrease, short deadline tasks start to miss their deadlines due to the delay in server capacity being supplied unless server capacities are greatly inflated, and hence the overall schedulability of the system decreases.

7. 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 fixed priority 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, even if the cache is flushed after each global context switch.

We presented a number of approaches to calculate inter-component CRPD in a hierarchical system with varying levels of sophistication. We also showed that when taking inter-component CRPD into account, minimising server periods does not maximise schedulability. Instead, the server period must be carefully selected to minimise inter-component CRPD while still ensuring short deadline tasks meet their time constraints.

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 i.e. 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 and 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.

Previous works by Lipari and Bini [19] and Davis and Burns [11] have investigated how to select sever parameters. In future, we intend to extend this work to find optimal server parameter settings taking into account inter-component CRPD. Lunniss *et al.* [20] 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. Finally, we intend to apply the approaches to a real system, in order to show how the different techniques compare using results obtained via measurement.

ACKNOWLEDGEMENTS

This work was partially funded by the UK EPSRC through the Engineering Doctorate Centre in Large-Scale Complex IT Systems (EP/F501374/1), the UK EPSRC funded MCC (EP/K011626/1), the European Community's ARTEMIS Programme and UK Technology Strategy Board, under ARTEMIS grant agreement 295371-2 CRAFTERS, COST Action IC1202: Timing Analysis On Code-Level (TACLe), and the European Community's Seventh Framework Programme FP7 under grant agreement n. 246556, "RBUCE-UP".

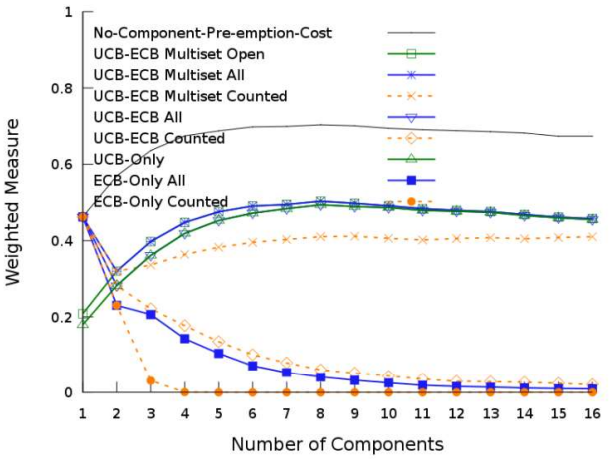

Figure 6. Varying the number of components from 1 to 16, while keeping the number of tasks in the system fixed.

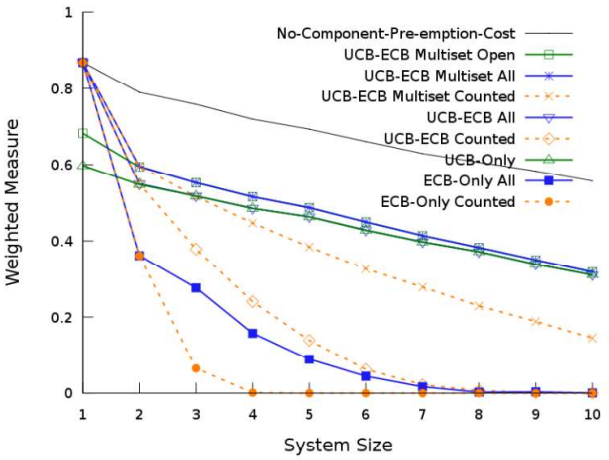

Figure 7. Varying the system size from 1 to 10. An increase of 1 in the system size relates to introducing another component adding 5 more tasks and increasing the cache utilisation by 2.

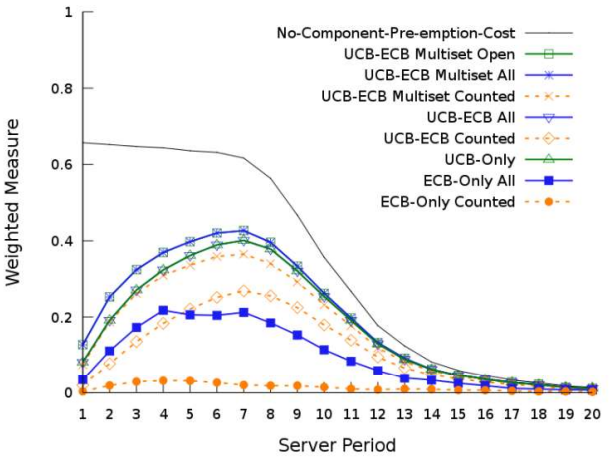

Figure 8. Varying the server period from 1ms to 20ms (fixed task period range of 10ms to 1000ms)

REFERENCES

[1] Altmeyer, S., Davis, R.I., and Maiza, C. 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 2011), 261-271.

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

[3] ARINC. *ARINC 651: Design Guidance for Integrated Modular Avionics*. Airlines Electronic Engineering Committee (AEEC), 1991.

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

[5] Åsberg, M., Behnam, M., and Nolte, T. An Experimental Evaluation of Synchronization Protocal 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 2013).

[6] Audsley, N. C., Burns, A., Richardson, M., and Wellings, A.J. Applying new Scheduling Theory to Static Priority Preemptive Scheduling. *Software Engineering Journal*, 8, 5 (1993), 284-292.

[7] Bastoni, A., Brandenburg, B., and Anderson, J. Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability. In *Proceedings of OSPERT* (Brussels, Belgium 2010), 33-44.

[8] Behnam, M., Shin, I., Nolte, T., and Nolin, M. 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)* (2007), 279-288.

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

[10] Busquets-Mataix, J. V., Serrano, J. J., Ors, R., Gil, P., and Wellings, A. 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)* (1996), 204-212.

[11] Davis, R. I. and Burns, A. An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-emptive Systems. In *Proceedings 16th International Conference on Real-Time and Network Systems (RTNS)* (Renne, France 2008), 19-28.

[12] Davis, R. I. and Burns, A. Hierarchical Fixed Priority Pre-emptive Scheduling. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)* (2005).

[13] Davis, R. I. and Burns, A. Resource Sharing in Hierarchical Fixed Priority Pre-Emptive Systems. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)* (Rio de Janeiro, Brazil 2006), 257-270.

[14] Davis, R. I., Zabos, A., and Burns, A. Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems. *IEEE Transactions on Computers*, 57, 9 (September 2008), 1261-1276.

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

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

[17] Kuo, T-W. and Li, C-H. A Fixed Priority Driven Open Environment for Real-Time Applications. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)* (Madrid, Spain 1998).

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

[19] Lipari, G. and Bini, E. A Methodology for Designing Hierarchical Scheduling Systems. *Journal of Embedded Computing*, 1, 2 (December 2005), 257-269.

[20] Lunniss, W., Altmeyer, S., and Davis, R. I. Optimising Task Layout to Increase Schedulability via Reduced Cache Related Pre-emption Delays. In *In proceedings of the International Conference on Real-Time Networks and Systems (RTNS)* (Pont à Mousson, France 2012), 161-170.

[21] Lunniss, W., Altmeyer, S., Lipari, G., and Davis, R. I. *Accounting for Cache Related Pre-emption Delays in Hierarchical Scheduling*. Technical Report YCS-2014-491 Available from http://www-users.cs.york.ac.uk/~wlunniss/, University of York, York, 2014.

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

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

[24] Saewong, S., Rajkumar, R., Lehoczky, J., and Klein, M. Analysis of Hierarchical Fixed Priority Scheduling. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS)* (Vienna, Austria 2002), 173-181.

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

[26] Staschulat, J., Schliecker, S., and Ernst, R. Scheduling Analysis of Real-Time Systems with Precise Modeling of Cache Related Preemption Delay. In *In Proceedings 17th Euromicro Conference on Real-Time Systems (ECRTS)* (Balearic Islands, Spain 2005), 41-48.

[27] Tan, Y. and Mooney, V. Timing Analysis for Preemptive Multitasking Real-Time Systems with Caches. *ACM Transactions on Embedded Computing Systems (TECS)*, 6, 1 (February 2007).

[28] Watkins, C. B. and Walter, R. Transitioning from Federated Avionics Architectures to Integrated Modular Avionics. In *Proceedings of the 26th IEE/AIAA Digital Avionics Systems Conference (DASC)* (2007).

[29] Xu, M., Phan, L. T.X., Lee, I., Sokolsky, O., Xi, S., Lu, C., and Gill, C. Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)* (Vancouver, Canada 2013).