# FOREWORD

## Message from the Program Chairs

It gives us great pleasure to introduce this volume that includes the Proceedings of the 5th Real-Time Scheduling Open Problems Seminar (RTSOPS 2014). This volume represents the continued openness in the Real-Time Systems research community to share and discuss unsolved problems concerning real-time scheduling theory.

This 5th edition of the seminar series is co-located with the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014) that is being held in Madrid, Spain. The day-long seminar features 8 open problems distributed over four sessions. Each session included two papers and collaboration time. All participants are invited to interact in groups and tackle the presented problems. The hope is that we make headway in solving the problems and that more complete problem definitions and solutions will emerge as a result of the discussions initiated during RTSOPS 2014.

We would like to thank the generosity of the Program Committee for their time and attention to detail that helped us assemble the program for the day. This year's Program Committee included:

- Sebastian Altmeyer (U. Amsterdam, Netherlands)
- Bjorn Brandenburg (MPI/SWS, Germany)
- Arvind Easwaran (Nanyang TU, Singapore)
- Vincent Nelis (ISEP/IPP, Portugal)
- Harini Ramaprasad (Southern Illinois University, USA)

We are also grateful to the RTSOPS Steering Committee for their feedback and advice. The RTSOPS Steering Committee includes

- Robert Davis (University of York, UK)
- Liliana Cucu-Grosjean (INRIA, France)
- Nathan Fisher (Wayne State Univ., USA)
- Gerhard Fohler (TU Kaiserslautern, Germany)

This program would not have been possible with the efforts and support of the ECRTS 2014 organizing committee and the General Chair, Juan de la Puente.

We hope you enjoy RTSOPS 2014 and that it is a productive event.
Marko Bertogna & Sathish Gopalakrishnan
*RTSOPS 2014 Program Chairs*

# TABLE OF CONTENTS

# Rare events and worst-case execution times

Cristian Maxim*,**, Liliana Cucu-Grosjean* and Benoit Triquet**
* firstname.lastname@inria.fr, AOSTE team, INRIA Paris-Rocquencourt
** firstname.lastname@airbus.com, Airbus, Toulouse

## 1 Motivation

During the last years the arrival of multi-core processors or many-core processors as well as the increased complexity of programs have made more difficult the estimation of the worst case execution times (WCETs) of programs. The existing methods may produce estimates that are too pessimistic for some systems. As result new analyses based on probabilities and statistics have appeared to cope with this complexity by taking into account the fact that large values of WCET may have low probability of appearance.

The first paper introducing probabilistic distributions for the description of execution times of tasks had associated to large values of execution times low probabilities [7] as illustrated in Figure 1. Different papers propose since methods to obtain such distributions. In [3] the authors provide a framework for obtaining the probabilistic execution times (pETs) of a program. Another method for estimating a pWCET bound in the presence of permanent faults in instruction caches was introduced in [6]. Papers like [4, 9] propose the estimation of pWCET using extreme value theory. Such theory is applied in [2] to platforms with randomized timing behavior and an associated avionics case study is presented in [8]. Only for this type of architecture, to our best knowledge, it is provided a proof that a large value of an execution time of a program is a rare event [1].

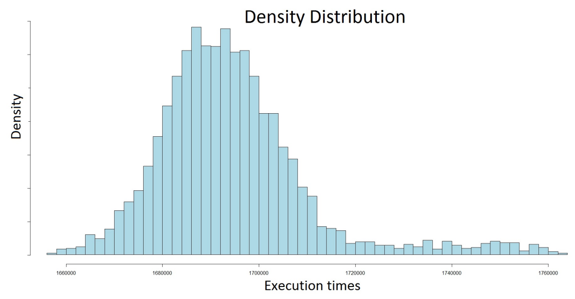


Figure 1: Distribution of execution times

## 2 Open Problem

In practice, it is noticeable that the higher the measured execution time is, the smaller its probability of occurrence is. In reality, the WCET is not easy to measure, and the analysis tools can either overestimate the WCET (static analysis), or underestimate it (taking in consideration only measurements), or predict it with a certain probability of occurrence (measurement-based probabilistic timing analyses). Figure 2 shows a description of the currently common accepted relation between observed execution times, WCET, etc [5].

As stated in the introduction associating low probability of appearance to large values of pETs was proved valid in the context of cache randomized architectures. One would expect to have **higher** probability of appearance for large values of pETs on existing real-world deterministic architectures (from which the vast majority are deterministic), but how one would prove it? For cache randomized architectures the proof was built using static probabilistic analysis and this does not seem to be trivial for any architecture. In conclusion our open problem is

**How do we prove that large values of pETs are rare events for real-world programs executed on existing deterministic architectures?**

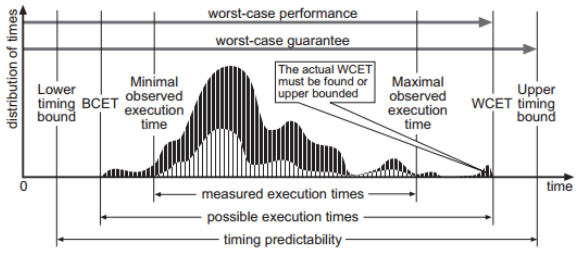


Figure 2: Commonly accepted relation between possible execution times

## References

[1] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. D. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. Proartis: Probabilistically analyzable real-time systems. *ACM Trans. Embedded Comput. Syst.*, 12(2s):94–114, 2013.

[2] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *the 24th Euromicro Conference on Real-time Systems (ECRTS)*, 2012.

[3] L. David and I. Puaut. Static determination of probabilistic execution times. In *the Euromicro Conference on Real-Time Systems(ECRTS)*, 2004.

[4] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium (RTSS01)*, 2001.

[5] Andreas Ermedahl and Jakob Engblom. Execution time analisis for embedded real-time systems, January 2007.

[6] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In *the 21st International Conference on Real-Time and Network Systems (RTNS) RTNS*, 2013.

[7] T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu, and J.S Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 1995.

[8] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quinones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *the 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2013.

[9] L. Yue, I. Bate, T. Nolte, and L. Cucu-Grosjean. A new way about using statistical analysis of worst-case execution times. *ACM SIGBED Review*, September 2011.

# Challenges with Probabilities in Response-Time Analysis of Real-Time Systems

Thomas Nolte*†, Meng Liu*, Björn Lisper*
*MRTC / Mälardalen University, Västerås, Sweden
†Software Systems / ABB Corporate Research, Västerås, Sweden

**Abstract**

In this paper we present and discuss some of the key open problems and challenges with using probabilities in Response-Time Analysis (RTA) of ”real” real-time systems, i.e., challenges inherent in the real software to be analyzed.

## I. Introduction

The Worst-Case Response-Time (WCRT) of a task is the maximum time after activation within which the task will finish its execution. The WCRT is constructed by the Worst-Case Execution Time (WCET) of the task itself, as well as the interference of other higher priority tasks, along with potential blocking by lower priority tasks if blocking is allowed by the task model. The interference of other higher priority tasks is constructed using the higher priority tasks' WCET and period. Hence, the WCET of tasks in a real-time system is central in deriving the WCRT of a particular task. Usually the WCET is derived using static analysis of the source code in a context of a specific processor on which the task is to execute on. The context is needed to get the correct timing characteristics.

Now let's consider that the probability of the WCET to actually happen is rather low. If we further consider that the tasks are independent from each other, then the probability of the actual WCRT to happen will be derived by multiplication of the corresponding WCET probability of the task under analysis along with the respective WCET probability of all instances of interfering and blocking tasks executing during the busy period constituting the response time. Even with a fairly high probability of the WCET to happen it is easy to see that the probability of the WCRT to happen quickly becomes very small.

Probabilistic WCRT has the potential to avoid over-allocation of system resources such as processor capacity which, if applied in RTA for real systems, increases resource efficiency and decreases hardware cost. Hence, it is tempting to make use of the probability in the context of WCRT, such that overly pessimistic (sufficiently unlikely) WCRT values can be disregarded. Instead, using a threshold to derive more realistic (sufficiently accurate) analytical response time values would be more applicable.

In search of the holy grail of probabilistic real-time analysis techniques a large number of approaches for using probabilities in real-time analysis theory have been developed over the years. However, most of these approaches are developed around a set of simplifying assumptions restricting the problem and unfortunately also restricting the usefulness of the approach in the setting of a real system.

In this paper we try to go to the bottom of the challenges with using probabilities in RTA for real-time systems. We identify a set of key challenges inherent in such analysis, and we discuss the implication of these challenges in this context of related work. Finally, we point towards the future with an attempt to see how probabilities can be used in the context.

## II. Challenges

Below we outline 6 key challenges in achieving useful probabilities in RTA. Note that when doing WCET analysis for WCRT analysis, it is easier to capture the exact worst-case related to each challenge not having to keep track of the different states outlined below. However, when going for a probabilistic RTA, a more detailed probabilistic execution time is needed along with various state information in order to produce meaningful probabilistic response times. For each challenge outlined below we describe the known assumptions made to address the challenge. Note that it is possible to construct a system and software taking one or more challenge's assumptions into consideration, however it can be concluded from complying with these assumptions that they make the analysis less useful for application to real systems.

### *Challenge 1 – SOFTWARE INTERNAL STATE*

The software internal state will affect the exact execution time of a particular execution instance of a task. Internal variables of the task may contain data that affect the exact software code to be executed and how, e.g., which software parts to execute and for how many iterations, which in turn will affect the exact execution time of a particular instance of task execution. Known assumptions made to solve the challenge:

A1.1 Forbid tasks to have internal states, i.e., each activation of a task is independent from other activations of the same task and therefore the task's execution time can be modeled as independent from the software internal state.

A1.2 Make sure that the internal state of a task will not influence its execution time.

*Challenge 2 – SOFTWARE EXTERNAL STATE*

Sharing of resources among the tasks will affect the exact execution time of a task instance. If critical sections are protected by locks, protocols for synchronization among real-time tasks can be used. There exist analysis for the worst case behavior of such protocols, however the exact execution time of a particular task execution instance is not known as this depends on how and when the other tasks sharing the same resource will execute. Known assumptions made to solve the challenge:

A2.1 Forbid synchronization and sharing of data among tasks, i.e., all tasks must be independent from each other.

A2.2 Make sure that the execution time of a task will not depend on effects related to shared data.

*Challenge 3 – SOFTWARE ARCHITECTURE STATE*

Due to usage of services and drivers located together with the task on a processor or core of a multi-core, e.g., a particular driver or service that is used by several tasks possibly also resident outside of the processor hosting the task under analysis, the exact execution time of a task instance will depend on the usage of these services and drivers. Note that it may be very difficult to characterize the exact behavior of service and driver usage by external clients compared to internal clients. Known assumptions made to solve the challenge:

A3.1 Avoid usage of shared services and drives in the software architecture.

A3.2 Enforce predictability of shared services and drivers, e.g., by hosting them in predictable containers such as servers.

*Challenge 4 – HARDWARE ARCHITECTURE STATE*

The hardware state will affect the execution time of the software executing on the hardware. The hardware state is affected by all software that is executing on the hardware. For example, the contents of caches will have an impact on the time that it takes to execute instructions and manipulate data. If two processor cores are sharing a cache, a task running on one core may interfere with the tasks running on the other core even though they are otherwise independent from each other. Known assumptions made to solve the challenge:

A4.1 Reset the hardware state before execution of a task instance, and to not allow for any preemption of the task before it has finished its execution.

A4.2 Disregard for the effect of hardware related interference. Depending on what hardware is used, the effect of the hardware internal state on the execution time of a task could be so small that it could be incorporated into, e.g., the execution time of the task, and therefore disregarded.

A4.3 Make the hardware to behave random, and therefore the effects of it can be modeled accurately.

*Challenge 5 – STATE OF THE ENVIRONMENT*

The state of the environment in which the task is executing may affect the exact execution time of a particular task execution instance. For example, a software tracking obstacles may have its execution time depend on the number of obstacles that it is currently tracking. Another example is the speed of a vehicle that may affect the execution time of a control software running in the vehicle, e.g., where the speed generate data to be processed by a task instance. Known assumptions made to solve the challenge:

A5.1 Disregard the environment from the analysis. In the context of a worst-case analysis this is a common solution, however if we would like to derive a correct distribution of execution- and response-times, the environment's affect on the execution time should be dealt with.

A5.2 Make sure that the implementation of the software in the task will not vary its execution time depending on the state of the environment, e.g., it is possible to implement (at least some) algorithms to have constant execution time.

## III. Discussion and Open Problems

We have contributed to the state-of-the-art in probabilistic and statistical analysis of real-time systems, e.g., [**?**], [**?**]. Most related work, including our much of our own, disregard either all or most of these challenges, i.e., they can be considered as open problems, in particular the combination of solutions to several of the challenges. Typically related work tries to address at most one of the challenges. We believe that addressing all challenges 1-5 is required for a general applicability of probabilistic real-time analysis in the context of "real" real-time systems, i.e., systems that are not subject to too many restricting assumptions. Hence, given current state-of-the-art, we have a lot of significant research challenges to explore in the upcoming years.

## References

[1] G. A. Kaczynski, L. Lo Bello, and T. Nolte, "Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems," in *12th IEEE Intl. Conference on Emerging Technologies and Factory Automation (ETFA'07)*, Sep. 2007, pp. 101–110.

[2] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A statistical response-time analysis of real-time embedded systems," in *33rd IEEE Real-Time Systems Symposium (RTSS12)*, December 2012, pp. 351–362. [Online]. Available: http://www.es.mdh.se/publications/2618-

# On the application of Static Probabilistic Timing Analysis to Memory Hierarchies

Benjamin Lesage[1], David Griffin[1], Robert I. Davis[1], Sebastian Altmeyer[2]
[1] University of York, UK / [2] University of Amsterdam, The Netherlands
{benjamin.lesage,david.griffin,rob.davis}@york.ac.uk, altmeyer@uva.nl

## I. Introduction

The temporal validation of a real-time system must ensure that critical tasks meet their deadline, even in the worst-case scenarios. Traditionally, both the deadline and worst-case execution time of a task are expressed as absolute bounds. Such absolutes may lead to resource over-provisioning for the sake of system validation, as they may include unlikely events with a granularity beyond the industry requirements. Probabilistic worst-case execution times (pWCET) fit industrial requirements for the validation of critical real-time systems as they provide bounds on the worst-case execution time of applications as well as their exceedance probability when run. Different static probabilistic timing analyses (SPTA) [2], [3], [1] have been proposed and extended to derive the probabilistic worst-case execution time (pWCET) of a task in the presence of randomised caches.

However, these methods only apply to single level caches and fail to capture the behaviour of complex memory hierarchies. Such hierarchies are common even in simple architectures as they bridge the gap between fast processors and relatively slower main memories. In the context of a memory hierarchy, multiple cache layers are linked together. They are traversed in order from the processor to the main memory, and a request is served by the first cache level where the target block is found. Therefore, the position of a requested block in the hierarchy defines the latency of a memory request, and the behaviour of subsequent requests.

Static analyses have been proposed to upper-bound the contribution of cache hierarchies on tasks' execution times; however, these approaches focus on deterministic policies [4], [6]. Deterministic policies do not lend themselves to the expression of worst-case timing estimates as exceedance functions, and the resulting absolute bounds may be pessimistic. The prior methods rely either on the classification of the access behaviour of each access with regards to each cache level for an incremental analysis level by level [4], or a unified model of the whole hierarchy [6].

## II. SPTA for Single Caches

Two complementary families of SPTA approaches, contention and collection, have been defined for single caches implementing an evict-on-miss random cache replacement policy. Under such a policy, when a requested block is absent from the cache, a line is randomly selected for eviction and the requested block, fetched from memory, replaces its contents. Each line has the same probability $\frac{1}{N}$, depending on the associativity $N$ of the cache, to be selected for eviction, hence minimising the dependencies of the cache behaviour on access history.

Contention-based methods approximate the Probability Mass Function (PMF) of each access, i.e. the probability of the access to suffer a cache hit or a memory hit latency, such that their convolution results in a sound timing estimate. The convolution operator imposes restrictions on the derived hit probabilities. The PMF of an instruction must lower-bound its cache hit probability, and hence its execution time. As should the convolution of different PMF lower-bound the execution time of the corresponding sequence of accesses [1]. The former is ensured by the definition of a lower bound on hit probability based on the reuse distance of accesses, the maximum number of evictions from the previous access to the same block. The latter is guaranteed in the most recent approach [1] using cache contention, which relates to the number of hits between two accesses, to ensure that the lines that are considered as hits fit into the cache.

Collection approaches instead rely on the approximation of a sound subset of the possible cache states at runtime, alongside their corresponding execution time distributions. Compared to contention-based methods they offer an increased precision at the cost of higher complexity. The complexity can be controlled by focusing the analysis on a subset of memory blocks deemed relevant, which in turn introduces uncertainty in the model and reduces its precision.

## III. Impact of memory hierarchies on SPTA

Consider an $M+1$ level memory hierarchy, including the main memory. A memory request is first processed by the lowest cache level the $L1$ cache. In the case of a miss, it is then transferred to the next layer and so on until it can be served. The only guarantee after the access is the presence of the block in the $L1$ cache. The contents of each layer after an access depends on the management policy. Each policy may introduce new dependencies between instructions and cache levels. Hence, the introduction of multiple levels in the memory hierarchy challenges the assumptions behind existing SPTA techniques.

For both contention and collection approaches, the hierarchy management policies need to be studied i) to prevent the explosion of the problem, e.g. from a single state an eviction on $L1$ and $L2$ produces $N_1 \times N_2$ states assuming $N_i$ is the associativity of cache level $i$, and ii) capture scenarios for which convolution produces only valid sequences of events, e.g. a hit in the $L2$ cache may depend on a prior access being a miss in $L1$.

## A. Inclusive cache hierarchy

An inclusive cache hierarchy enforces the inclusion of the contents of a cache in the higher levels, e.g. each block in the L1 cache must also be in the L2 cache. When a block is evicted from a high cache level, it is removed from lower levels, a so-called *inclusion victim*. The higher level caches must be able to hold all lines in the respective lower levels.

After a memory request, the block is guaranteed to be present in all levels inclusive of the first one. The reuse distance of an access can be estimated from the closest prior reference to the same block. Further, because of *inclusion victims*, a miss on cache level $L$ contributes to the reuse distance of all lower levels. Thus an access for $LM$ may contribute to the reuse distance of accesses on the $L1$,..., $LM-1$ caches. This is a worst-case scenario assuming a memory hit. The contribution to the reuse distance on the $L1$ of a guaranteed level $L$ hit cannot exceed $L-1$. But randomised caches hardly provide such guarantees on hits beyond the first level. Deterministic architectures rely on history-dependent policies, not amenable in the context of SPTA, to reduce the impact of *inclusion victims* [5].

## B. Exclusive cache hierarchy

An exclusive cache hierarchy maximises the use of the cache space. Every cache line in the hierarchy holds a different block. A memory block can only appear in one cache level. Upon eviction from a cache level $L$, the victim is inserted in cache level $L+1$. The requested data is only inserted in the $L1$ cache upon a miss. Therefore, all caches in an exclusive hierarchy must use the same line size to allow for line swapping. Although this is not mandatory, an equal number of sets across the different levels eases the implementation and the analysis.

An exclusive cache hierarchy may be modelled as a single cache which size equals the size of the hierarchy. Given a hierarchy of LRU caches, the logical age of a block then defines the cache level in which it resides and therefore its access latency [4]. A similar model can be applied to collection analyses for randomised caches, but the depth of a block in the hierarchy needs to be upper-bounded as it defines the latency of accesses. If different levels of the hierarchy use a different number of cache sets, the unified contents model may not hold.

The reuse distance could also be computed using this unified contents representation. A block can only be evicted from its current cache level if an access is served by a higher level than the one where it resides. A victim block is inserted in the higher cache, where it is less sensitive to evictions. This reinforces the need for a per-level expression of the reuse-distance, even though it is likely to be the same for most levels. Capturing guaranteed hits, requests which do not reach higher levels is complex for layers beyond the first.

## C. Non/Mostly-inclusive cache hierarchy

Neither the exclusion or inclusion restriction applies on a non-inclusive hierarchy. Upon a miss on a cache level, the request is passed to the next level until it can be served. An access guarantees the insertion of the data in all the levels where it could not be found, up to the first hit in the hierarchy. This is the simplest implementation of a cache hierarchy. From the analysis point of view, it means that guarantees about the presence of data in cache after an access only exist for the first level, or if misses can be guaranteed. Deriving sound PMF requires a lower bound on the miss probabilities of accesses, e.g. an access can only be a $L3$ hit if it misses on all lower levels.

There is no correlation between the different levels. Contrary to the other policies it is unsafe to consider that a piece of data is in higher levels than the one it might reside in. This has been identified in the deterministic case, where the insertion of a block in a level which is not guaranteed to be accessed results in optimistic timing estimates. Instead, when the access to a level cannot be guaranteed, a collection approach can insert a placeholder for the data in cache, such that further accesses to the same block do not cause additional evictions, but cannot assume hits on this data. An important property to capture for memory blocks is the lowest and highest level where they might reside in the hierarchy. These two values bound the cache levels where the eviction of blocks is bound to occur but not the insertion.

It is currently an open problem how to effectively and efficiently perform SPTA for any of the above multilevel memory hierarchies using random replacement caches.

## Acknowledgment

This work was funded by COST Action IC1202 (TACLe), and the EU FP7 Integrated Project PROXIMA (611085).

## References

[1] Sebastian Altmeyer and Robert I Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In $17^{th}$ *DATE*, 2014.

[2] Francisco Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. Proartis: Probabilistically analysable real-time systems. *Transactions on Embedded Computing Systems*, 2013.

[3] Robert I Davis, Luca Santinelli, Sebastian Altmeyer, Claire Maiza, and Liliana Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. In *25th ECRTS*, 2013.

[4] Damien Hardy and Isabelle Puaut. Wcet analysis of instruction cache hierarchies. *Journal of Systems Architecture*, 57(7), 2011.

[5] Aamer Jaleel, Eric Borch, Malini Bhandaru, Simon C. Steely Jr., and Joel Emer. Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (tla) cache management policies. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 151–162, 2010.

[6] T. Sondag and H. Rajan. A more precise abstract domain for multi-level caches for tighter wcet analysis. In *IEEE 31st RTSS*, 2010.

# Progress on static probabilistic timing analysis for systems with random cache replacement policies

Sebastian Altmeyer, University of Amsterdam, altmeyer@uva.nl
Liliana Cucu-Grosjean, INRIA Paris-Rocquencourt, liliana.cucu@inria.fr
Robert I. Davis, University of York, rob.davis@york.ac.uk
Benjamin Lesage, University of York, benjamin.lesage@york.ac.uk

## I. Original Problem Statement

Real-time systems such as those deployed in space, aerospace, automotive and railway applications require guarantees that the probability of the system failing to meet its timing constraints is below an acceptable threshold (e.g. a failure rate of less than $10^{-9}$ per hour). Advances in hardware technology and the large gap between processor and memory speeds, bridged by the use of cache, make it difficult to provide such guarantees without significant over-provisioning of hardware resources. The use of deterministic cache replacement policies means that pathological worst-case behaviours need to be accounted for, even when in practice they may have a vanishingly small probability of actually occurring. The use of cache with random replacement policies [3] can negate the effects of pathological worst-case behaviours while still achieving efficient average-case performance, hence providing a way of increasing guaranteed performance in hard real-time systems.

The timing behaviour of programs running on a processor with a random cache replacement policy can be determined using Static Probabilistic Timing Analysis (SPTA). SPTA computes an upper bound on the probabilistic Worst-Case Execution Time (pWCET) in terms of an exceedence function, which gives the probability, as a function of all possible values for an execution time budget $x$, that the execution time of the program will not exceed that budget on any single run. SPTA [5] requires a probability function that can be used to compute an estimate of the probability of a cache hit for each memory access. This probability function is *valid* if it provides a lower bound on the probability of a cache hit. As shown last year at RTSOPS 2013 [4], the only valid cache-hit probability known by then is given as follows:

$$\hat{P}^D(k) = \begin{cases} \left(\frac{N-1}{N}\right)^k & N > k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $N$ denotes the associativity of the cache and $k$ the reuse distance, i.e., the number of intervening memory accesses that could cause an eviction, since the memory block was last accessed. All other estimations of the hit-probability [7, 6] that had been proposed by then have been refuted as they may lead to optimistic results. The complexity of deriving a sound estimate of the hit-probability is caused by the dependency of the current event of a cache hit or miss on the history of prior events; caused by the finite size of the cache. In Equation (1), this dependency is accounted for by setting the probability of a cache hit to zero in cases where the reuse distance exceeds the associativity; which results in a large over-approximation even for simple access sequences. The open problem presented in last year's RTSOPS [4] was thus: *how to improve upon the simple SPTA analysis?*

## II. Correctness Conditions and Optimality

Instead of immediately answering the open problem, we tried to learn from the failed approaches to improve upon Equation (1) and identified the correctness conditions [2] that any sound approximation of the cache-hit probability must fulfil. Sound in this context means that for any sequence of cache accesses $[e_1, \dots, e_n]$, the approximation $\hat{P}$ complies with two constraints: (**C1**) it does not over-estimate the probability of a cache hit, and (**C2**) the value obtained from convolution of the approximated probabilities for any subset of a trace $T$ describing the probability that all elements in the subset are a hit, is at most the precise probability of such an event occurring:

**C1** $\forall e \in [e_1, \dots, e_n]\colon P(e^{hit}) \geq \hat{P}(e^{hit})$,

**C2** $\forall E \subseteq [e_1, \dots, e_n]\colon P\left(\bigwedge_{e \in E} e^{hit}\right) \geq \prod_{e \in E} \hat{P}(e^{hit})$.

Using these soundness conditions, we have been able to clearly identify why former approaches [7, 6] failed and we have been able to show that Equation (1) is not only correct, but also optimal with respect to the limited information it uses: any cache-hit probability that only uses the associativity and the reuse distance is either at most as precise as Equation (1) or optimistic. Due to space limitation, we refer to [1] for the proof of optimality.

## III. Using other information

The negative result that we can not improve the existing cache-hit probability by using the same information also gives the key to providing better bounds: we have to include additional information which is not yet taken into account.

### A. Stack Distance

$\hat{P}^D(k)$ can be pessimistic in the commonly observed case of sequences with repeated accesses (e.g. loops). For example, the trace $a, b, c, d, c^1, d^1, c^1, d^1, a^7, b^7$ repeats the accesses $c, d$ three times within the reuse distance of the final accesses to $a$ and $b$. Assuming an associativity of 4, then $\hat{P}^D(k)$ gives zero probability of a cache hit for these accesses, since their reuse distance exceeds the associativity of the cache. However, it is possible for the cache to contain all four distinct memory blocks $a, b, c, d$ accessed in this sequence, and so a zero value for the probability of a cache hit for the final accesses to $a$ and $b$ is pessimistic.

Let $\Delta$ be the *stack distance* of element $e_l$, i.e., the total number of pair-wise *distinct* memory blocks that are accessed within the reuse distance $k$ of element $e_l$. The maximum number of distinct cache locations loaded during the reuse distance of $e_l$ is upper bounded by $\Delta$, hence it follows that a lower bound on the probability that $e_l$ will survive all of the loads and remain in the cache is given by:

$$\hat{P}^A(\Delta, k) = \begin{cases} \left(\frac{N-\Delta}{N}\right) & (N > \Delta) \wedge (k \neq \infty) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

We note that $\hat{P}^A(\Delta, k)$ and $\hat{P}^D(k)$ are incomparable, yet both give valid lower bounds on the probability of a cache hit. We thus may use the maximum of them to compute an improved lower bound that dominates each individually.

## B. Cache Contention

Equation (1) and Equation (2) both provide a tight lower bound on the probability of a cache hit, but are imprecise even for simple access sequences. If we consider for instance a random cache with associativity 4 and the following access sequence, $a, b, c, d, f, a^4, b^4, c^4, d^4, f^4$ all accesses are considered cache misses. The reason for this is that for each of the last five accesses, the probability of a cache hit is set to 0 to ensure correctness with respect to condition **C2**, i.e, that the probability of the last five access all being hits is zero. However, this can also be ensured by considering the probability of a cache hit for the preceding accesses. To this end, we define the concept of the *cache contention* of a memory block $e_l$ which denotes the number of memory accesses within the reuse distance of $e_l$ that potentially contend with $e_l$ for space in the cache. We only need to set the probability of a cache hit for an access $e_l$ to zero when the cache contention is greater than or equal to the associativity $N$.

$$\hat{P}^N(e_l^{hit}) = \begin{cases} 0 & con(e_l, T) \geq N \\ max\left(\hat{P}^A(\Delta, k), \left(\frac{N-1}{N}\right)^k\right) & \text{otherwise} \end{cases} \tag{3}$$

Conceptually, the cache contention assumes that each access within the reuse distance of $e_l$ that has been assigned non-zero probability of being a hit as requiring its own separate location in the cache. Due to space limitation, we refer to [1] for the exact definition of the cache contention.

## IV. Collecting Semantics and Combined Approach

An orthogonal approach to compute the pWCET is to enumerate all possible cache states and the associated probabilities. As this solution is computationally intractable, we have developed a combined approach with scalable precision: The idea is to use the precise approach for a small subset of *relevant* memory blocks, while using the imprecise approach for the remaining blocks. So, instead of enumerating all possible cache states, we abstract the set of cache states and focus only on the $m$ most important memory blocks, where $m$ can be chosen to control both the precision and the runtime of the analysis. In this way, we effectively reduce the complexity of the precise component of the analysis for a trace with $l$ distinct elements from $2^l$ to $2^m$ (typically with $m \ll l$). We again refer to [2] for the details of this approach.

## V. Open Problems and Future Work

This progress report presents the solutions to one of last year's open problems: *how to improve upon the simple SPTA analysis?* A first negative result, namely that the original hit probability can not be improved without additional information, has led us towards (i) the discovery of alternative approaches to bound cache-hit probability that rely on additional information such as the stack distance and the cache contention and (ii) the development of an orthogonal approach that relies on complete, or partial enumeration of the cache contents. As according to George Bernard Shaw *science never solves a problem without creating ten more*, the recent advancements lead to new, open problems. Foremost, how to extend the analysis to control-flow graphs and how to select the relevant memory blocks for the combined approach.

## Acknowledgements

This work was funded by COST Action IC1202 (TACLe), and the EU FP7 Integrated Project PROXIMA (611085).

## References

[1] Sebastian Altmeyer and Robert I. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. Technical Report YCS-2013-487, University of York, 2013. Available from http://www.cs.york.ac.uk/ftpdir/reports/2013/YCS/487/YCS-2013-487.pdf.

[2] Sebastian Altmeyer and Robert I. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *DATE 2014*, page tbp, 2014. Available from http://www.cs.york.ac.uk/ftpdir/reports/2013/YCS/487/YCS-2013-487.pdf.

[3] Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery D. Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. Proartis: Probabilistically analyzable real-time systems. *ACM Trans. Embedded Comput. Syst.*, 12(2s):94, 2013.

[4] Robert I. Davis. Improvements to static probabilistic timing analysis for systems with random cache replacement policies. In *RTSOPS 2013*, pages 22–24, 2013.

[5] Robert I. Davis, Luca Santinelli, Sebastian Altmeyer, Claire Maiza, and Liliana Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. In *ECRTS 2013*, pages 129–138, 2013.

[6] Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones, and Francisco J. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE 2013*, pages 513–518, 2013.

[7] Shuchang Zhou. An efficient simulation algorithm for cache of random replacement policy. In *NPC 2010*, pages 144–154, 2010.

# How to deal with control-flow information in parallel real-time applications?

José Carlos Fonseca, Vincent Nelis, Gurulingesh Raravi and Luís Miguel Pinho
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: {jcnfo,nelis,guhri,lmp}@isep.ipp.pt

## I. Motivation for a new parallel application model

Traditional sequential programming models struggle to harness the immense processing capabilities offered by the new massively parallel architectures. It is also widely recognized that coarse-grained thread-based parallel programming models scale poorly due to load balancing difficulties. As a result, a shift in software development towards fine-grained task-based parallel programming paradigms is currently under way. For the real-time scheduling community, this paradigm shift entails the adoption of new application models to represent *precedence constraints* and *synchronization points* among all the computing units (referred to as *sub-tasks* hereafter).

Current models typically represent a parallel application as a single *graph*. Each *node* of the graph models a sequential sub-task and the *edges* that connect the nodes model all kinds of functional and/or non-functional dependencies between these sub-tasks. Every sub-task (node) is typically characterized by (at least) a WCET estimation and by some additional parameters such as a best-case/average-case execution time estimation, a memory footprint, and an input and output list of parameters. The edges may represent functional or non-functional precedence constraints, as in the fork/join task model [1], or explicit data dependencies as in the synchronous parallel task model [2] and the parallel DAG model [3]. In addition, every edge may also be annotated with additional information/constraints on the dependency between the two nodes that it connects. However, *none of these models can capture the control-flow information (such as conditional execution of code) of the applications that they model.* Consequently, in such models, *all* the nodes in the graph *must* execute each time the application is run. As a result, a simple application (see Fig. 1a) that executes in parallel either (as a result of an if-statement) four instances of a function $B$ or two instances of a function $C$, cannot be properly modelled by using any of the parallel application models proposed so far in the real-time literature. All the existing models are limited by their implicit assumption that all the nodes must execute each time the application is run. Due to this limitation, these models:

**Case 1**: model function $A$ as a node connected to four nodes $B$ and two nodes $C$, which are in turn connected to node $D$ (see Fig. 1b). The resulting graph is then composed of 8 nodes that are all assumed to be executed each time the application is run. This leads to an obvious over-approximation of the maximum workload[1] (denoted $W$) of the application, which makes the higher level analyses (like the schedulability analysis) more pessimistic.

**Case 2**: consider only the "worst-case execution flow" of the application and model that single flow as a graph of sub-tasks that must all execute. However, it is not trivial to determine the worst-case execution flow of a parallel application. Assuming an infinite number of cores, the WCET of the application is reached by executing the flow (i.e. the graph of sub-tasks) of maximum "critical path length" (denoted CP). In our example, such a WCET is reached by taking the "else" flow depicted in Fig. 1d which has the longest critical path length ($= 1 + 5 + 2 = 8$). However, this flow is not necessarily the one causing the maximum interference to the other applications of the system. Assuming that our example application has the highest priority in a system with 6 cores, although its WCET is reached by executing the "else" flow, this flow will occupy at most two cores at the same time and has a maximum workload of $W = 13$. On the other hand, the "if" flow depicted in Fig. 1c is shorter (CP $= 7$) but it may cause more interference on the lower priority applications as it will demand more processing resources ($W = 19$) from four cores at the same time. Therefore, when the schedulability of all the applications in the system has to be assessed, it is a non-trivial task to identify these so-called application worst-case execution flow and to the best of our knowledge, this problem is still an open one.

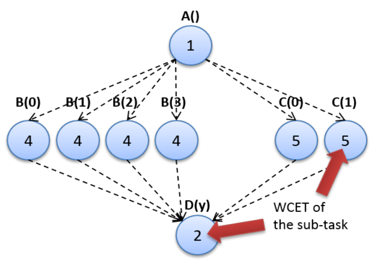
Our research identifies the need to support modelling of such applications for which each run may execute a different set of sub-tasks and all these sub-tasks may have different dependencies from one run to another. The main difference with the existing parallel application models is that we intend to represent an application as *a set of DAGs* where each DAG models a single execution flow of the application. We believe that such a model is more accurate and efficient than the state-of-the-art as it does *not* create a single DAG by cross-cutting the structures of multiple execution flows with different parameters (for instance, we will model the "if" and "else" flows of Fig. 1c and Fig. 1d as two separated DAGs instead of modelling the application by using the graph of Fig. 1b). We also believe that one of the first computation phases of the currently-available WCET analysis tools [4], during which the control flow graph is reconstructed by parsing the code of the application, can be adapted to our application model such that the tool will identify every feasible execution flow. However, instead of modelling an execution flow as a simple sequence of instructions or basic blocks (as the tools currently do), a flow will be modelled as a DAG of sub-tasks with data dependencies between them. Further, we also believe that the same tools may be adapted to derive various useful information on every execution flow such as its maximum workload, its "critical path length", its maximum concurrency level, etc.

From a real-time perspective, this new model poses serious challenges, especially when trying to compute the response time of an application scheduled conjointly with other applications. With this model, it must be verified that *every* feasible execution flow (which is now a DAG) of the application under analysis completes before the application deadline, and this must hold

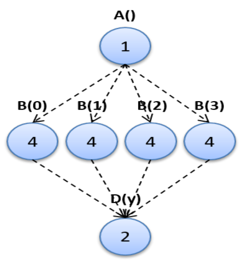
[1]The maximum workload is defined as the maximum amount of work that the application can impose on the system, i.e. it is the sum of the WCET of all its nodes.

```
x = A();
y = 0;
if (x > 1) {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 4; i++)
      y += B(i);
} else {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 2; i++)
      y += C(i);
}
z = D(y)
```

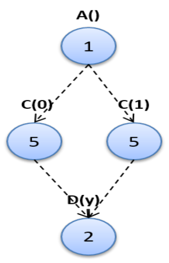
**(a)** Example of a parallel application using Open MP.



**(b)** Complete graph of the application. $W = 29$ and $\mathrm{CP} = 8$.



**(c)** "If" execution flow. $W = 19, \mathrm{CP} = 7$.



**(d)** "Else" execution flow. $W = 13, \mathrm{CP} = 8$.

true considering all possible interference patterns from the other applications. Knowing that this interference can greatly vary depending on which execution flow is taken by these other applications (some of these flows may execute for a very long time but using only a few cores whereas others may use many cores and execute for a very short time), the interference analysis could consider every combination of execution flows from every application in the system, but this would be prohibitively expensive in terms of the computation time.

## II. The Envisioned Approach

The objective of this work is to come up with a framework to enable the computation of the response time of every application in the system in polynomial time. Our approach intends to compute a single DAG of independent periodic servers for each application in the system, and define a mapping rule to decide which ready sub-task must be assigned to which server at run-time. These servers and the mapping rule must be defined such that, for any execution flow taken by the application at run-time, enough cpu-budget is provided to the sub-tasks of the currently executed flow so that the application completes by its deadline and respects all the precedence constraints. Specifically, our envisioned solution will work as follows:

**Step 1)** We will propose an efficient mapping rule to arbitrate the assignment of the sub-tasks that are ready-for-execution to the running servers.

**Step 2)** *For each execution flow (DAG of sub-tasks) of the application under analysis*, we will compute a graph of servers where each node of the graph (i.e. each server) has a cpu-budget and each edge models a precedence constraint between two servers. This graph of servers together with the mapping rule defined in Step 1, will provide the following guarantee at run-time: no matter how the servers are scheduled on the cores, *as long as their precedence constraints are not violated*, the total budget provided by *the $k$'th job of every periodic server* ($\forall k > 0$) — and the way this budget is distributed between the sub-tasks by the mapping rule — provides enough cpu-budget for all the sub-tasks of the currently-taken execution flow to complete.

**Step 3)** All the graphs of servers computed in Step 2 for every execution flow of the application will be merged into a single graph of servers. That resulting graph will be defined such that it preserves the guarantee defined in Step 2, i.e. it provides enough cpu-budget to the sub-tasks of any execution flow.

**Step 4)** The resulting graph of servers computed in Step 3 will be converted into a set $\tau$ of asynchronous periodic constrained-deadline independent real-time tasks by using techniques such as [5], where each real-time task $\tau_i$ is defined by four parameters: an offset $O_i$, an execution time (cpu-budget) $C_i$, a period $T_i$, and a deadline $D_i \leq T_i$.

**Step 5)** Finally, by repeating Step 1 to Step 4 for each parallel application in the system, we will end up with several sets of real-time tasks (one set for each application). We can then group all these task sets in a super-set $\overline{\tau}$ of asynchronous periodic constrained-deadline independent real-time tasks. This task model has been widely studied in the past and many efficient scheduling algorithms have been proposed. Our solution will then investigate which of these scheduling algorithms can successfully schedule the resulting set $\overline{\tau}$ by using their respective schedulability tests.

## Acknowledgments

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project(s) FCOMP-01-0124-FEDER-037281 (CISTER), FCOMP-01-0124-FEDER-020447 (REGAIN); by the European Union, under the Seventh Framework Programme (FP7/2007-2013), grant agreement nr. 611016 (P-SOCRATES).

## References

[1] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *the 31st RTSS*, 2010, pp. 259–268.
[2] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *the 32nd RTSS*, 2011, pp. 217–226.
[3] S. K. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *the 33rd RTSS*, 2012, pp. 63–72.
[4] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.
[5] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Techniques optimizing the number of processors to schedule multi-threaded tasks," in *the 24th ECRTS*, July 2012, pp. 321–330.

# Preemption costs integration invalidates popular feasibility conditions of asynchronous periodic tasks scheduled with EDF

Thomas Chapeaux[1], Laurent George[2] and Joël Goossens[1]
[1]Université Libre de Bruxelles, Belgium
[2]University of Paris-Est / LIGM, France

## 1 Introduction

In this paper we revisit the problem of uniprocessor real-time scheduling of asynchronous periodic tasks for EDF scheduling when preemption cost in taken into account in the feasibility conditions. We show that the classical feasibility interval $[0, O_{\max} + 2H]$ [2] where $O_{\max}$ is the maximal offset value and $H$ is the hyperperiod of the tasks (the least common multiple of the periods) can be insufficient to check the schedulability of EDF schedule.

We consider a task model where each task $\tau_i$ is described by a tuple $(O_i, C_i, D_i, T_i)$ where $O_i$ is the arrival time of the first job (or *offset*), $C_i$ is the execution time of each job, $D_i$ is the relative deadline of each job, and $T_i$ is the exact time between two consecutive job arrivals.

For fixed priority scheduling and EDF scheduling with this task model, if there is a time $t'$ such that $\forall t \geq t'$ we have that the state of the execution is the same at $t$ and at $t - k$ (where $k$ is a constant superior to 0), then we call the interval $[0, t']$ the *transient phase* and the rest of the execution the *periodic phase* (composed of a repeating pattern of size $k$ called the *Period*). By definition, in order to determine if a system is schedulable by EDF, it is sufficient to simulate its execution over the transient phase (i.e. until the periodic phase is reached). It has been shown that for systems such that $\sum_i \frac{C_i}{T_i} \leq 1$ (a necessary condition of feasibility), the execution reaches its periodic phase at the latest at $t = O_{\max} + 2H$ [3], [1], with a Period of size $H$.

This model assumes preemption costs to be negligible. When this is not the case, a common approach to integrate them is to inflate the tasks WCET by some value (e.g. the cost of one preemption multiplied by the maximal number of preemption per job). This is very pessimistic as the number of preemptions for each job of the same task can vary greatly, so this approach often leads to resource waste.

In this paper, we want to integrate the preemption costs into the model on a per-preemption basis. Therefore systems are now described as a set of tasks (as before), with the addition for each task $\tau_i$ of a parameter $\alpha_i$, which is an upper bound of the cost associated with the preemption of one of its jobs. During the execution, when a previously preempted job of task $\tau_i$ is re-activated, it first has to execute a (preemptive) recovering interval during $\alpha$ time units before resuming its actual execution. In the case of fixed priority scheduling, the feasibility interval $[0, O_{\max} + 2H]$ remains valid in this model [4]. We now show that this is not true for EDF scheduling.

## 2 Size of the transient phase and Period

We show that both the transient phase and the Period may be larger than the results presented in the previous section when the preemption costs are taken into account, by giving the example of the system described by $\tau_1 = (0, 4, 10, 10)$, $\tau_2 = (3, 11, 30, 30)$, $\tau_3 = (13, 4, 30, 30)$ and $\alpha_1 = 0$, $\alpha_2 = \alpha_3 = 2$.

If we execute this system under EDF scheduling and with the preemption costs taken into account (as seen in Figure 1), we see that the system does not reach its periodic phase until $t = O_{\max} + 6H = 193$, where it is at the same state as in $t = O_{\max} + 4H = 133$. This means that not only the transient phase is longer than $O_{\max} + 2H$, the Period is also longer than $H$.

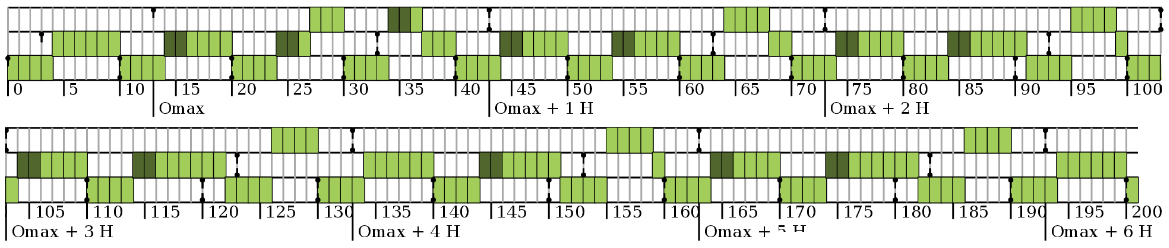


Figure 1: Longer transient phase and Period. Darker schedule units represent recovering intervals caused by a previous preemption.

## 3 Non-sufficiency of the feasibility interval

Now consider the same system, but with $D_2 = 28$. This system has the exact same EDF schedule (as seen in Figure 2) and thus does not miss any deadline before $O_{\max} + 2H$, however it does miss a deadline at $t = 121 > O_{\max} + 3H$. This invalidates the previous result stating that it is sufficient to simulate EDF for the first $O_{\max} + 2H$ time units to determine schedulability by EDF. This is a consequence of the results of Section 2, but this example confirms that a deadline miss can indeed occur after $O_{\max} + 2H$.

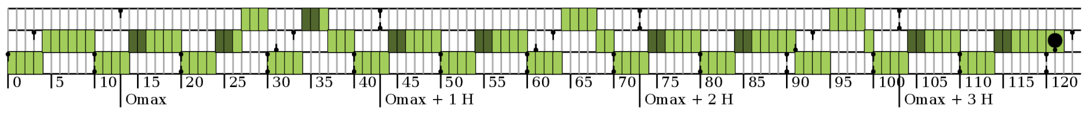


Figure 2: Deadline miss after $O_{\max} + 2H$.

## 4 Conclusion

We have shown that when the preemption costs are integrated into the feasibility analysis, the classic results on the length of the feasibility interval and the periodic behaviour for EDF are invalidated. The determination of the minimal length of the transient phase for EDF in that context remains an open problem.

## References

[1] B. Bado, L. George, P. Courbin, J. Goossens, *A semi-partitioned approach for parallel real-time scheduling.*. Proceedings of the 20th International Conference on Real-Time and Network Systems, RTNS'2012, Pont à Mousson, France, p. 151–160, ACM, 2012.

[2] S. Baruah, L. Rosier, R. Howell, *Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor*. The Journal of Real-Time Systems, Kluwer Academic Publishers, p 301-324, Vol. 2, n. 4, 1990.

[3] A. Choquet-Geniet,E. Grolleau, *Minimal schedulability interval for real-time systems of periodic tasks with offsets*. Theoretical Computer Science, Vol. 70, 2004.

[4] P. Meumeu Yomsi, Y. Sorel, *Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems*. 19th Euromicro Conference on Real-Time Systems, ECRTS'07, Pisa, Italy, 2007.

# Open Problems in Multi-Modal Scheduling Theory for Thermal-Resilient Multicore Systems

Nathan Fisher Masud Ahmed Pradeep M. Hettiarachchi

Department of Computer Science, Wayne State University, Detroit, Michigan, USA
{fishern, masud, pradeepmh}@wayne.edu

## I. Introduction

Recent research on thermal-aware real-time systems has focused upon developing design frameworks that ensure predictable timing behavior for systems executing in dynamic thermal environments. For such dynamic environments, a *thermally-resilient real-time system* [4] must adapt both its timing requirements and thermal-dissipation rates in response to a changing environmental temperature. On the hardware side, thermal dissipation rates might potentially be reduced via dynamic voltage/frequency scaling (DVFS) and/or dynamic power management (DPM) techniques such as putting the CPU into low-power/idle states. On the software side, the adaptation of timing requirements can be achieved through *multi-modal tasks systems*, providing the means to change task timing parameters (e.g., execution requirement, period, deadline, etc.) and/or to add/drop tasks from the system. The ultimate goal for thermal-resilient real-time systems is the capability of a real-time system designer to accurately predict (at design time) what real-time mode can be supported for any given environmental temperature.

Our initial work on the development of a design framework for thermal-resilient real-time systems focused on single-core systems (see Hettiarachchi et al. [4]). To achieve both adaptivity and predictability in our design framework, we introduced the concept of a *real-time performance mode* which explicitly coupled hardware and software modes together. The hardware mode was characterized by the *periodic resource model* [12] which can quantify the execution supply of a non-continuously executing resource (e.g., a processor that switches from an active to an inactive state periodically to change the thermal dissipation of the CPU). Each software mode was characterized by a *sporadic task system* [7]. In complementary work, we developed single-core schedulability analysis for the real-time performance mode model [2] specified by this periodic-resource/sporadic-task-system combination; thus, the resulting framework is able to provide hard-real-time guarantees.

The above design framework has subsequently been extended to multicore platforms [5]. However, a restrictive limitation is that currently the framework supports only partitioned scheduling of tasks for each mode. The reason for this limitation is that ***there is a fundamental gap in multi-mode scheduling theory for multicore platforms with non-continuous execution patterns***. As our thermal-resiliency framework (and thermal-aware real-time systems, in general) requires schedulability analysis for dynamic processing platforms that utilize DVFS/DPM, it is important that new analysis techniques be developed that will permit these systems to implement a larger set of the global-scheduling algorithms. In this abstract, we will briefly describe the real-time performance mode (i.e., HW/SW-coupled multi-modal) model in the context of global scheduling, outline important scheduling theory questions for this model, and review relevant prior research.

## II. Real-Time Performance Mode Model for Globally-Scheduled Systems

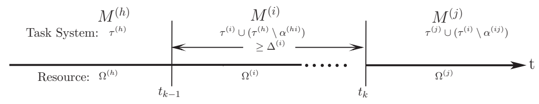
We consider a system with $m$ number of identical CPU cores. The set of real-time performance modes is denoted $\mathcal{M}$. Each real-time performance mode $M^{(i)} \in \mathcal{M}$ is characterized by a three-tuple $(\tau^{(i)}, \Omega^{(i)}, \Delta^{(i)})$. The first parameter $\tau^{(i)}$ denotes a sporadic task system with $n_i$ tasks. Each sporadic task $\tau_j^{(i)} \in \tau^{(i)}$ is characterized by a three-tuple $(e_j^{(i)}, d_j^{(i)}, p_j^{(i)})$ where $e_j^{(i)}$ is the *worst-case execution requirement*, $d_j^{(i)}$ is the *relative deadline*, and $p_j^{(i)}$ is the *minimum inter-arrival separation parameter*. The first job of $\tau_j^{(i)}$ may arrive at any time after mode $M^{(i)}$ is activated.

The second parameter of the real-time performance mode $\Omega^{(i)}$ denotes the resource parameters (i.e., the execution supply guaranteed by the underlying multicore platform). In our previous work, we used a periodic resource $(\Pi^{(i)}, \Theta^{(i)})$ to represent the periodic active/idle times of each core. The term $\Theta^{(i)}$ represents the minimum resource active durations that is guaranteed in periodic $\Pi^{(i)}$ length intervals when the system is operating in mode $M^{(i)}$ mode. However, this model is not expressive enough or flexible enough for global scheduling. Thus, for globally-scheduled systems, $\Omega^{(i)}$ may be expressed in a more appropriate compositional scheduling model. Potential candidate models for $\Omega^{(i)}$ include the *parallel-supply function* (PSF) abstraction [1] and the *multiprocessor periodic resource* (MPR) [11] model. As an example, in the MPR model, $\Omega^{(i)}$ can be characterized by a three-tuple $(\Pi^{(i)}, \Theta^{(i)}, m^{(i)})$ where the MPR guarantees a total of $\Theta^{(i)}$ units of execution in each $\Pi^{(i)}$-length period with maximum level of parallelism of $m^{(i)}$ (i.e., at any given time there are at most $m^{(i)}$ processors executing tasks of $\tau^{(i)}$).

The third parameter $\Delta^{(i)}$ denotes the minimum amount of time that the processor will remain in mode $M^{(i)}$. The motivation for this parameter comes from discrete control where changes in a system (plant) occur only at discrete intervals of time. In our thermal-resilient design framework, the mode changes occur only at these sampling intervals. In fact, it is convenient and often the case that $\Delta^{(i)}$ can be set as a multiple of the resource period $\Pi^{(i)}$.

This work has been supported in part by the US National Science Foundation (Grant Nos. CNS-0953585 and CNS-1205338).

§**Mode-Change Semantics.** A *mode-change request* $\mathsf{mcr}_k \equiv (M^{(i)}, M^{(j)}, t_k)$ consists of *transition time* ($t_k$), the *old mode* $M^{(i)}$ executing prior to $t_k$, and the new mode $M^{(j)}$ executing after $t_k$ (where $i, j \in \{1, \ldots, q\}$). We assume that if $i < j$ then $\mathsf{mcr}_i$ occurs prior to $\mathsf{mcr}_j$. For $\mathsf{mcr}_{k-1} \equiv (M^{(h)}, M^{(i)}, t_{k-1})$ and $\mathsf{mcr}_k \equiv (M^{(i)}, M^{(j)}, t_k)$, the difference $t_k - t_{k-1}$ must be at least $\Delta^{(i)}$ to ensure the minimum activation time for mode $M^{(i)}$. Figure 1 illustrates a mode-change-request sequence.


$M^{(h)}$
$M^{(i)}$
$M^{(j)}$
Task System: $\tau^{(h)}$
$\tau^{(i)} \cup (\tau^{(h)} \setminus \alpha^{(hi)})$
$\tau^{(j)} \cup (\tau^{(i)} \setminus \alpha^{(ij)})$
$\geq \Delta^{(i)}$
t
Resource: $\Omega^{(h)}$
$\Omega^{(i)}$
$\Omega^{(j)}$
$t_{k-1}$
$t_k$
Fig. 1. Mode-change sequence.

Tasks may be divided into groups based on their importance at the time of the mode change request $\mathsf{mcr}_k \equiv (M^{(i)}, M^{(j)}, t_k)$. (See Real and Crespo [10] for complete taxonomy). Some important tasks may need to continue to execute without being affected by the mode change request. We call these tasks *unchanged tasks* denoted by $\tau^{(ij)}$. Some less important tasks may be removed from the system immediately at the time of the mode change request. We call these tasks *aborted tasks* and denote them by $\alpha^{(ij)}$. For some tasks, immediate termination may leave the system in an inconsistent state; we call such tasks *finished tasks* and characterize them as being members of the set $\tau^{(i)} \setminus (\alpha^{(ij)} \cup \tau^{(ij)})$. We allow a job from a finished task at the time of a mode change request to complete its remaining execution. Given the above definitions, we may distinguish three phases with respect to a mode-change request $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ (from the previous request $\mathsf{mcr}_{k-1} = (M^{(h)}, M^{(i)}, t_{k-1})$):

1) $[t_{k-1}, t_k)$: jobs of $\tau^{(i)}$ are executed upon $\Omega^{(i)}$;
2) $[t_k, t_{k+1})$: incomplete jobs of $(\tau^{(i)} \setminus \alpha^{(ij)})$ at $t_k$ and jobs of $\tau^{(j)}$ execute upon $\Omega^{(j)}$;
3) $[t_{k-1}, t_{k+1})$: unchanged tasks ($\tau^{(ij)}$) will act independent of mode change request.

## III. Open Problems & Related Work

Informally, our main open problem statement is simple to state: *given an $m$-core platform and multi-modal system $\mathcal{M}$ determine whether all tasks of every modes will always meet all deadlines under any possible legal mode-change sequence*. However, there are many dimensions to this problem that make it possible to define a "family" of related subproblems. In particular, we may consider the following different aspects of the system:

**D1** *Global Scheduling Algorithm*: schedulability analysis is needed for both global earliest-deadline-first and fixed-priority scheduling algorithms to be applicable for real systems that implement the thermal-resiliency framework.

**D2** *Multiprocessor Resource Model*: As mentioned in the previous section, we can consider different models such as PSF or MPR for expressing the $\Omega^{(i)}$ parameter.

In addition to the schedulability analysis problem, an important variant problem is determining the minimum allocation for each mode to maintain schedulability. Some open questions for the allocation problem are: 1) What is the most appropriate definition of "minimum allocation" for thermal-resilient systems? (e.g., if $\Omega^{(i)}$ is an MPR, is minimizing the sum of the capacities a good objective to decrease total thermal dissipation?); and 2) How do we efficiently deal with the combinatorial nature of the problem? (I.e., there might be numerous combinations of capacities that are schedulable).

§**Relevant Prior Work.** Extensive research effort has been focused on developing multi-modal schedulability analysis for multicore platforms (e.g., see [3], [6], [8], [9]). To the best of our knowledge, each of these prior works assumes the underlying processing cores execute at a fixed speed/rate. However, we hope that these multi-modal results for fixed-rate multicore platforms might be extendable to platforms with DVFS/DPM capabilities to solve the open problems posed in this abstract. A starting point might be trying to extend the multi-modal analysis for uniform multiprocessors [13] to real-time performance modes.

## References

[1] E. Bini, M. Bertogna, and S. Baruah. Virtual multiprocessor platforms: Specification and use. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 437–446, Dec 2009.
[2] N. Fisher and M. Ahmed. Tractable real-time schedulability analysis for mode changes under temporal isolation. In *Proceedings of the 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTImedia)*. IEEE Computer Society, October 2011.
[3] J. Goossens and P. Richard. Partitioned scheduling of multimode multiprocessor real-time systems with temporal isolation. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 297–305, 2013.
[4] P. M. Hettiarachchi, N. Fisher, M. Ahmed, L. Y. Wang, S. Wang, and W. Shi. The design and analysis of thermal-resilient hard-real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 67–76, 2012.
[5] P. M. Hettiarachchi, N. Fisher, and L. Y. Wang. Achieving thermal-resiliency for multicore hard-real-time systems. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*, pages 37–46, July 2013.
[6] J. Lee and K. G. Shin. Schedulability analysis for a mode transition in real-time multi-core systems. In *Proceedings of the 34th IEEE Real-Time Systems Symposium*, pages 11–20, Dec 2013.
[7] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.
[8] V. Nelis, B. Andersson, J. Marinho, and S. M. Petters. Global-edf scheduling of multimode real-time systems considering mode independent tasks. pages 205–214, 2011.
[9] V. Nelis, J. Goossens, and B. Andersson. Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, July 2009.
[10] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, 26:161–197, March 2004.
[11] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 181–190, July 2008.
[12] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3), April 2008.
[13] P. M. Yomsi., V. Nelis, and J. Goossens. Scheduling multi-mode real-time systems upon uniform multiprocessor platforms. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–8, Sept 2010.

# Optimal Speedup Bound for 2-Level Mixed-Criticality Arbitrary Deadline Systems

Xiaozhe Gu, Arvind Easwaran
Nanyang Technological University, Singapore
Email: guxi0002@e.ntu.edu.sg, arvinde@ntu.edu.sg

## I. Background

For safety-critical embedded systems, an increasing trend is to integrate multiple independent functionalities on a single shared computing platform; this trend is evident in industry-driven initiatives and standards such as ARINC653 [1] and IMA and AUTOSAR [2] in automotive. However among all the functionalities in a system, only some functionalities are critical for the correctness of the system and need to be certificated by statutory certification authorities (*CAs*) and the rest of the functionalities will not influence the safety of the system to the same extent. Under normal conditions, all tasks providing different functionalities will receive enough processing resource. But more critical tasks should be given preference if something abnormal happens in the system (i.e. when tasks exceed their expected worst-case execution times (WECTs) [3] which in fact is highly unlikely to happen). Such systems comprised of tasks of different criticalities are called mixed-criticality (*MC*) systems.

In many previous studies of mixed-criticality real-time systems, the model proposed by Vestal is widely used [3]. In this model, a task can be specified as $\tau_i = (T_i, L_i, \mathcal{C}_i, D_i)$, where $T_i$ denotes minimum separation between job releases, $L_i$ denotes criticality level based on the task's importance to the system, $\mathcal{C}_i = \mathcal{C}_i^1, \mathcal{C}_i^2, \cdots, \mathcal{C}_i^{max}$ is a list of WCET values measured with different degrees of confidence (here $\mathcal{C}_i^j$ denotes $\tau_i$'s WCET when the system is at criticality level $j$), and $D_i$ ($\leq T_i$) denotes $\tau_i$'s deadline. Jobs of $\tau_i$ are released with a separation of at least $T_i$ time units, and each job needs to execute for no more than $C_i^j$ time units within $D_i$ time units from its release time when the system is at criticality level $j$. Once any task executes more than $C_i^j$ time units then the system criticality level increases to $j+1$ and all tasks with criticality lower than $j+1$ are no longer required to finish. But tasks with criticality higher than $j$ will need to execute for $C_i^{j+1}$ time units within $D_i$. According to the relationship between $T_i$ and $D_i$, the system can be classified into three categories: constrained deadline task systems (for any $\tau_i$, $D_i \leq T_i$), implicit deadline task systems (for any $\tau_i$, $D_i = T_i$) and arbitrary deadline task systems (for any $\tau_i$, $D_i > T_i$). We will focus on 2-level mixed-criticality task systems on uniprocessor platforms in this paper where $\tau_i$ is either a low-criticality task or a high-criticality task. Hence $L_i \in \{L, H\}$ (L denotes low criticality and H denotes high criticality) and $C_i \in \{C_i^L, C_i^H\}$.

## II. previous work

Many mixed-criticality scheduling algorithms like EDF-VD [4], AMC [5], GREEDY [6] and ECDF [7] have been developed based on the model proposed by Vestal [3], and different metrics like run-time complexity, schedulability, feasibility are used to evaluate them. Among multiple metrics, the speedup bound is a very important one because the problem of scheduling mixed-criticality workloads is proven to be NP-hard [8], and a speedup bound can show how good this algorithm can be compared to an optimal algorithm in a very intuitive way.

*Definition 1:* The speedup bound of an algorithm denotes the minimal processor speed the algorithm needs to successfully schedule any mixed-criticality real-time system which is feasible on an unit speed uniprocessor platform.

Previously, it has been shown in the paper [8] that $\frac{1+\sqrt{5}}{2}$ is a bound of speedup for 2-level mixed-criticality constrained deadline systems. This speedup bound is based on a fixed-priority algorithm, namely OCBP [9], and is proved to be tight i.e., $\frac{1+\sqrt{5}}{2}$ is the best speedup bound that can be derived from OCBP. A better bound of $\frac{4}{3}$ is derived from a dynamic-priority algorithm, namely EDF-VD [4], but only valid for implicit deadline task systems. EDF-VD uses a deadline tightening strategy to deal with the most challenging problem that mixed-criticality systems face. This problem arises from the need to execute an additional demand of up to $C_i^H - C_i^L$ for each high-criticality task as system criticality changes, while task deadlines remain the same.

*Definition 2:* If the algorithm is optimal with regard to the speedup bound factor, then no non-clairvoyant scheduling algorithms can have a tighter speedup bound.
It proves that EDF-VD is optimal from the perspective of speedup bound by using an example taskset to show that no non-clairvoyant scheduling algorithm[1] can schedule the taskset on a processor with speed less than $\frac{4}{3}$, although the taskset is feasible on an unit speed processor [4]. So $\frac{4}{3}$ is the best speedup bound for any non-clairvoyant scheduling algorithm for 2-level mixed-criticality implicit deadline task systems on uniprocessor platform.

[1]Non-clairvoyant scheduling algorithms have no knowledge of when a job may trigger a mode-switch.

For 2-level mixed-criticality arbitrary deadline task systems, the best known speedup bound is $1+\frac{\sqrt{3}}{2}$, which is also derived from the algorithm EDF-VD [10]. An example is given to show that there exist feasible tasksets on an unit speed processor but cannot be scheduled by EDF-VD on a processor that is $1+\frac{\sqrt{3}}{2}-\varepsilon$ times fast, where $\varepsilon$ tends to 0. However, this only proves that $1+\frac{\sqrt{3}}{2}$ is a tight speedup bound for EDF-VD scheduling for 2-level mixed-criticality arbitrary deadline task systems, and it is possible that other algorithms may have a tighter speedup bound.

## III. Open Question

From the previous sections, the best known speedup bound for 2-level mixed-criticality constrained deadline task system is $\frac{1+\sqrt{5}}{2}$, and the best known speedup bound for 2-level mixed-criticality implicit-deadline and arbitrary-deadline task systems are $\frac{4}{3}$ and $1+\frac{\sqrt{3}}{2}$ respectively. As $\frac{4}{3}$ is already proved to be optimal from the perspective of speedup bound, the only remaining open problem in this area is that of finding optimal speedup bounds for constrained and arbitrary deadline task systems.

Note that an algorithm with better schedulability[2] is more likely to have a better speedup bound if it has one because it may need less extra speed to schedule feasible tasksets. Besides we should also note that to find the speedup bound of an algorithm, we should always consider the worst case scenario, meaning tasksets whose successful scheduling by the algorithm requires the maximum speedup. So we can try to derive better speedup bound from other algorithms that have better performance in terms of schedulability. However from the observation of existing speedup bounds, only simple algorithms[3] tend to have a speedup bound. For example, EDF-VD uses a very simple but almost unreasonable deadline tightening strategy: tightened deadline for all high-criticality tasks are proportional to their period $T_i$. In fact there exist many more reasonable deadline tightening strategies. For instance, one could assign tightened deadlines in proportion to the increase in execution requirement $\frac{(C_i^H - C_i^L)}{T_i}$, or efficient heuristic strategies like those in ECDF [7] and GREEDY [6]. Although ECDF and GREEDY have been shown to have better schedulability than EDF-VD, no speedup bounds are known for either of these approaches. This is mainly because they use demand bound functions (dbf) [11] (dbf is the maximum demand that the task can impose in any time interval of that length) in their schedulability tests, and dbfs are mathematically complex and generally not suitable for deriving speedup bounds.

Many mixed-criticality algorithms like GREEDY [6] or ECDF [7] use the strategy of tightened deadlines, but it looks unlikely that these algorithms with good performance but complex deadline tightening strategies would also have good speedup bounds. Maybe an alternate method is to use dbf to bound the demand of a taskset. Though there is no exact dbf for a mixed-criticality sporadic taskset, sufficient schedulability tests based on approximate dbf have shown good performance in in terms of schedulability [7]. In the paper [6], there were 2 dbfs, one for low-criticality mode and another for high-criticality mode. The pessimism in carry-over jobs was high and therefore the dbf approximation was pessimistic. This has been improved with ECDF [7] because the new dbf is a combined dbf for low-criticality mode and high-criticality mode. This means the load resulting from the new dbf will be more tight compared to the earlier separate dbf. This will help with a tighter load-based schedulability test and hence can lead to improved speedup bounds. With this test, it's possible to approximate the schedulability of an algorithm with more complex deadline tightening strategy. To find a speedup bound for dbf-based tests, we first need to approximate the dbf using simple linear upper bounds similar to ”load” used for multiprocessor tests [12]. We then need to derive load-based schedulability tests and corresponding deadline tightening strategies that would lead to efficient speedup bounds. Since load-based tests have resulted in good speedup bounds for multiprocessor scheduling algorithms [12], we believe that this direction of research has potential for mixed-criticality scheduling as well.

## Acknowledgements

This work was supported by MoE Tier-2 project no.: MOE2013-T2-2-029.

## References

[1] *”ARINC653 - An Avionics Standard for Safe, Partitioned Systems”*. Wind River Systems / IEEE Seminar, 2008.
[2] “Automotive Open System Architecture (AUTOSAR),” http://www.autosar.org/.
[3] S. Vestal, “Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance,” in *RTSS*, 2007.
[4] S. Baruah, V. Bonifaci, G. D”Angelo, H. Li, and A. Marchetti-Spaccamela, “The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems,” in *ECRTS*, 2012.
[5] S. Baruah, A. Burns, and R. Davis, “Response-Time Analysis for Mixed Criticality Systems,” in *RTSS*, 2011, pp. 34–43.
[6] P. Ekberg and W. Yi, “Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks,” in *ECRTS*, 2012, pp. 135–144.
[7] A. Easwaran, “Demand-based Scheduling of Mixed-Criticality Sporadic Tasks on One Processor,” in *RTSS*, 2013, pp. 78–87.
[8] S. Baruah, V. Bonifaci, G. D’Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, “Scheduling Real-Time Mixed-Criticality Jobs,” *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140–1152, 2012.
[9] H. Li and S. Baruah, “An Algorithm for Scheduling Certifiable Mixed-Criticality Sporadic Task Systems,” in *RTSS*, 2010, pp. 183–192.
[10] H. Li, “Scheduling Mixed-Criticality Real-Time Systems,” PHD thesis, 2013, in Chapter 5
[11] S. Baruah, A. Mok, and L. Rosier, “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor,” in *RTSS*, 1990, pp. 182–190.
[12] S. Baruah and N. Fisher, “Non-migratory feasibility and migratory schedulability analysis of multiprocessor real-time systems,” *Real-Time Systems*, vol. 39, no. 1-3, pp. 97–122, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11241-006-9009-7

[2]Schedulability here denotes the ability to successfully schedule a mixed-criticality feasible taskset.

[3]Simple algorithms here means either fixed priority algorithms (OCBP) or algorithms with simple deadline strategies (EDF-VD use a single scaling factor to set tightened deadlines).