Proceedings of the 8th Real-Time Scheduling Open Problems Seminar (RTSOPS)

held in conjunction with the 29th Euromicro Conference on Real-Time Systems (ECRTS)

Edited by Vincent Nelis[1] and Tam Chantem[2]
[1]*CISTER, ISEP, Portugal*
[2]*Virginia Tech, USA*

Dubrovnik, Croatia
June 27, 2017

Message from the Chairs

It gives us great pleasure to introduce this volume that includes the Proceedings of the 8th Real-Time Scheduling Open Problems Seminar (RTSOPS 2017). 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 8th edition of the seminar series is co-located with the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017) that is being held in Dubrovnik, Croatia. The day long seminar features 10 paper presentations and one keynote presentation distributed over 3 sessions, each one ending with an open collaborative session for participants to present new scheduling problems and discuss status of known problems on-the-spot.

Each session includes ample collaboration time, during which we encourage all participants 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 the workshop.

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. We are also grateful to the RTSOPS Steering Committee for their feedback and advice. This program would not have been possible without the efforts and support of the ECRTS 2017 organizing committee, in particular the General Chair Martina Maggio.

We invite all of you to join us in taking advantage of this excellent opportunity to learn and interact with our fellow colleagues. We hope you enjoy RTSOPS 2017.

Vincent Nelis[1] and Tam Chantem[2]
[1]CISTER, ISEP, Portugal
[2]Virginia Tech, USA
RTSOPS 2017 Program Chairs

RTSOPS 2017 Technical Program Committee

Robert Dick, University of Michigan, U.S.A.

Emmanuel Grolleau, Université de Poitiers, France

Luís Lino Ferreira, CISTER, ISEP, Portugal

Cong Liu, University of Texas, Dallas, U.S.A.

Damien Masson, ESIEE, Paris, France

Gang Quan, Florida International University, U.S.A.

Shangpin Ren, Illinois Institue of Technology

Xiaofeng Wang, University of South Carolina, U.S.A.

Dakai Zhu, University of Texas, San Antonio, U.S.A.

RTSOPS Steering Committee

Liliana Cucu-Grosjean, INRIA Paris, France

Nathan Fisher, Wayne State University, USA

Robert Davis, University of York, UK

Technical Program

On the Meaning of pWCET Distributions and their use in Schedulability Analysis

Robert I. Davis
University of York, UK
and Inria, France
Email: rob.davis@york.ac.uk

Alan Burns
University of York, UK
Email: alan.burns@york.ac.uk

David Griffin
University of York, UK
Email: david.griffin@york.ac.uk

Abstract

This short paper accompanies the keynote talk at RTSOPS 2017. It discusses the different meanings attached to probabilistic Worst-Case Execution Time (pWCET) distributions derived from Static Probabilistic Timing Analysis (SPTA) and Measurement-Based Probabilistic Timing Analysis (MBPTA). These different meanings relate to aleatoric variability (randomness in the systems and its environment) and epistemic uncertainty (lack of knowledge about the system) respectively. The different meanings have significant implications in terms of the valid use of pWCET distributions in probabilistic schedulability analysis.

I. Introduction

Verifying the timing correctness of real-time systems is typically a two step process.

- *Timing Analysis* is used to characterise the maximum amount of time which each software component or task can take to execute on the hardware platform. Typically, this is done by estimating, as a single value, an upper bound on the *Worst-Case Execution Time* (WCET).
- *Schedulability Analysis* is used to characterise the end-to-end response time of functionality involving one or more components, taking into account the way in which the components are scheduled, and any interference between them. Schedulability analysis typically makes use of WCETs to compute an upper bound on the Worst-Case Response Time (WCRT) which can then be compared to the deadline to determine timing correctness.

During the past decade the hardware platforms used, or proposed for use, in real-time embedded systems have become increasingly more complex. Architectures include advanced hardware acceleration features such as pipelines, branch prediction, out-of-order execution, caches, scratchpads, and multiple levels of memory hierarchy, as well as multi-core and many-core processors. These advances, along with increasing software complexity, greatly exacerbate the timing analysis problem. Most acceleration features are designed to optimise average-case rather than worst-case behaviour and can result in significant variability in execution times. This is making it increasingly difficult, if not impossible, to obtain tight WCET estimates from conventional static timing analysis methods that seek to provide a single absolute upper bound on the WCET. Further, increases in software and hardware complexity make it extremely difficult to ensure that the worst-case path(s) through a program have been exercised, and that the worst-case software and hardware states have been encountered in measurement-based analyses.

Probabilistic WCET analysis provides an alternative approach reflecting the fact that the absolute WCET of some software components running on advanced hardware platforms cannot be precisely determined. Instead of assuming a single absolute value for the WCET, probabilistic WCET analysis characterises the worst-case execution time of a component using a probability distribution. The precise meaning of this probabilistic Worst-Case Execution Time (pWCET) distribution is discussed in the next section. A probabilistic description of the worst-case execution time behaviour of a component can be used to estimate the probability that timing overruns may occur, and to size execution time budgets appropriately. The pWCET distribution can also be used via probabilistic schedulability analysis to upper bound the probability that deadlines may be missed.

Research into probabilistic WCET analysis can be classified into two main categories:

- *Analytical methods*: referred to as *Static Probabilistic Timing Analysis (SPTA)* [4], [7], [2], [1], [12]. SPTA is applicable when some part of the system or its environment contributes random or probabilistic timing behaviour, for example a random replacement cache. SPTA methods analyse the software, at both a high level (structural) and a low level (instructions), and use a model of the hardware behaviour to derive an estimate of worst-case timing behaviour, represented by a pWCET distribution, that is valid for any possible inputs, software states, hardware states[1], and paths through the code. SPTA does not execute the code on the actual hardware; rather it relies heavily on the model of the hardware being correct.
- *Statistical methods*: referred to as *Measurement-Based Probabilistic Timing Analysis (MBPTA)* [3], [10], [11], [6], [18], [17], [16], [14], [13]. MBPTA makes use of measurements (*observations*) of the overall execution time of the software

[1]Note any random variable, for example a random number generator within a random replacement cache, that gives rise to variation in timing behaviour is not included in these hardware states.

component, obtained by running it on the actual hardware, using test vectors i.e. inputs that exercise a relevant subset of the possible paths through the code, as well as the different software and hardware states that affect timing behaviour[2]. Rather than taking the maximum observed execution time and then adding some engineering margin, these methods use a statistical analysis of the observations based on *Extreme Value Theory* (EVT) to estimate the pWCET distribution.

II. Uncertainty and pWCET distributions

It is important to understand the precise meaning of a pWCET distribution since this impacts how such information can be used. In fact there are two subtly different meanings originating from SPTA and MBPTA, which is a potential source of confusion. We consider systems as having a *functional* behaviour i.e. what the system does in response to its inputs, and a *timing* behaviour i.e. how long it takes to respond to those inputs. Systems may have functional behaviour which is *deterministic*, in other words, given the exact same inputs, they will produce the exact same outputs. Functional behaviour may also be *non-deterministic*, for example a randomised search. Here we are concerned only with the timing analysis of software that has deterministic functionality. Timing behaviour may also be characterised as deterministic or it may depend on some element that can be characterised by a random variable, for example a random replacement cache. Hardware that supports deterministic timing behaviour is referred to as *time-predictable*, whereas hardware that deliberately introduces some random elements into the timing behaviour is referred to as *time-randomised*.

In general, uncertainty about the timing behaviour of a system can be classified into two categories:

- *Aleatoric variability* depends on chance or random behaviour within the system itself or its environment.
- *Epistemic uncertainty* is due to things that could in principle be known about the system or its environment, but in practice are not, because the information is hidden or cannot be measured or modelled.

While complex software running on advanced time-predictable hardware may in theory exhibit deterministic timing behaviour and therefore have a single absolute WCET associated with a particular set of inputs, software state, and hardware state, in practice this actual WCET cannot be determined and must therefore be estimated. Such an estimate is subject to epistemic uncertainty. In contrast, software running on simple time-randomised hardware exhibits aleatoric variability in its execution time. SPTA can be used to model aleatoric variability, but must deal with any epistemic uncertainty by upper bounding its effects in the model used. (For example if an instruction has a variable latency dependent on its operands, whose values are unknown, then SPTA can model that instruction as always assuming its worst-case latency). MBPTA can be used with systems that are characterised by either or both aleatoric variability and epistemic uncertainty.

As an example, it is instructive to consider a thought experiment involving two hypothetical systems. Both systems have 10 inputs which can take values in the range 1-6.

- **System A**: has two paths through the code. The first path is taken if the sum of the input values is odd, and takes 40 cycles to execute. The second path is taken if the sum of the input values is even, it has 10 instructions, each of which takes a random amount of time from 1-6 cycles to execute (independent of any other instruction or input value). Thus the overall execution time of this path resembles the total from rolling 10 fair dice.
- **System B**: has a single path, it uses a huge internal 10-dimensional array (with 6^{10} entries) that maps from the values of the 10 inputs to a delay. The values for the delays are the totals for each possible permutation of 10 dice rolls; however, they are randomly arranged in the array, and we do not necessarily know what that arrangement is. Further, half of the values have been set to 40 cycles; again, we do not necessarily know which ones. This system looks up its execution time from the table, using the input values, and executes in total for that amount of time.

Intrinsically, System A has only aleatoric uncertainty, while System B has only epistemic uncertainty.

Consider applying SPTA to System A. With an accurate model of the instruction timing behaviour, SPTA could be used to compute a pWCET distribution that upper bounds the timing behaviour of this system *irrespective* of its inputs.

In the context of SPTA, the meaning of a pWCET distribution can be defined as follows, building on the definition in [7]:

Definition 1: The pWCET distribution from SPTA is a tight upper bound[3] on all of the probabilistic execution time (pET) distributions that could be obtained for each individual combination of inputs, software states, and hardware states, excluding the random variables which give rise to variation in the timing behaviour. (Note, each individual pET distribution depends on the random variables, but not on the inputs or states, which are fixed in a particular combination).

Figure 1 illustrates an example pWCET distribution, showing WCET estimates $C1$ and $C2$ with different probabilities of exceedance. Also shown are the pET distributions for a few fixed combinations of inputs, software states, and hardware states. (Note this figure is only an example, it is not intended to resemble the pWCET for either System A or B).

[2]Exercising all possible paths and states is typically intractable.

[3]In the sense of the greater than or equal to operator defined on the 1 - CDF of the distributions [9].

In the absence of any random variables contributing to probabilistic timing behaviour, then the above definition of a pWCET distribution reduces to the familiar one for a single valued WCET obtained via conventional static WCET analysis. It is a tight upper bound on all the execution times that may be obtained for different combinations of inputs, software states, and hardware states.

If the random variables contributing to a probabilistic execution time behaviour are independent, then it follows that the pWCET distribution obtained by SPTA is independent with respect to any particular execution of that component. This is the case, since the pWCET distribution from SPTA upper bounds every pET distribution valid for a specific combination of inputs, software states, and hardware states. This has implications for the use of pWCET distributions, since they are independent they may be composed using basic convolution to derive probabilistic Worst-Case Response Time (pWCRT) distributions [8], [15], which can then be compared to the appropriate deadline to determine the probability of a deadline miss.

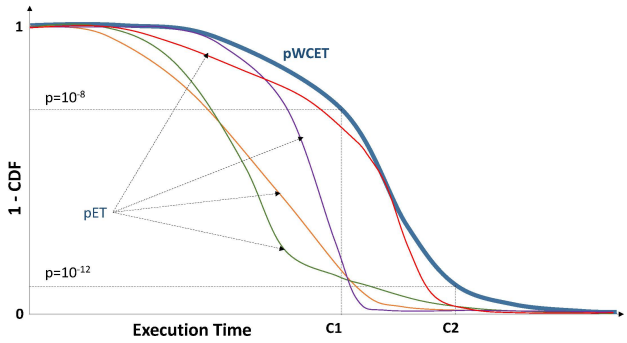


Fig. 1: Exceedance function (1-CDF) of pWCET distribution, and pET distributions for specific inputs.

Next, consider System B. Applying SPTA using a precise and detailed model of the software and hardware would result in a single WCET, since there are no random variables involved, and we assume no information about the frequency of any combination of input values. By contrast, if we apply MBPTA, then we can estimate the WCET; however, this estimate has *epistemic* uncertainty. There are things we do not know about the system when we consider it as a "black box", and we have only taken a sample of execution time observations, hence we cannot be 100% confident that our estimate is correct.

In the context of MBPTA, the meaning of a pWCET distribution can be defined as follows:

Definition 2: The pWCET distribution from MBPTA is a statistical estimate giving an upper bound p on the probability that the execution time of a component will be greater than some arbitrary value x, valid for any possible distribution of input values that could occur during deployment.

Thus the pWCET distribution characterises the probability $(1 - p)$ that the WCET of a component will be no greater than some arbitrary value x [5], or as noted by Edgar and Burns [10] the pWCET distribution reflects the *confidence* we have that the statement, *"the WCET does not exceed x for some threshold x"* is true.

We note that the definitions of a pWCET distribution originating from MBPTA and by SPTA are different. The definition from SPTA reflects aleatoric variability, while that from MBPTA reflects epistemic uncertainty. (Note there could also be an element of aleatoric variability from the system itself, for example if the hardware platform is time-randomised).

Since the pWCET definition from MBPTA reflects epistemic uncertainty, i.e. what isn't known about the system, then if it turns out that a WCET estimate x is exceeded, it is possible that it could be exceeded for *every* one of a number of runs of the component in a sequence, depending on the input values used. This is the case since the pWCET distribution effectively gives the probability that *at least one* run of the component has an execution time which exceeds x, but given that event, it provides no additional information about the execution times of individual runs.

For example, for System B, let us assume that MPBTA [6] estimates that there is a probability of 10^{-y} that the WCET exceeds x However, if that WCET estimate is exceeded, then it could be that it is exceeded *every* time the component runs, depending on the particular input values used. This has implications for how the pWCET distribution may be used in probabilistic schedulability analysis. Assuming a pWCET distribution derived via MBPTA where a WCET of x has an exceedance probability of 10^{-y}. We may only infer that N runs of the component have a probability of no more than 10^{-y} of exceeding a total execution time of Nx. Contrast this with a similar pWCET distribution derived via SPTA. In this case, assuming the aleatoric variability was due to independent random variables, then it would be valid to apply basic convolution to upper bound the overall execution time of N runs. This conclusion would not in general be sound with a pWCET distribution derived via MBPTA, due to its different meaning.

In the case of System A, the pWCET distribution from SPTA tells us that the probability that the execution time on any single run will exceed x is 10^{-y}. If we observe a value larger than x at some point in a large number of runs, then that is not in itself incompatible with the information that we have, which characterises aleatoric variability. By contrast, in the case of system B, the pWCET distribution from MBPTA gives us a *measure of confidence* that the WCET is no more than x. If we observe a value larger than x then that confidence falls to zero.

Another way of looking at the information provided by MBPTA, is to consider that among a universe of systems similar to system B that could produce the observations seen during analysis, then the probability that we are observing a system that has a WCET of more than x is estimated at 10^{-y}. Stated otherwise, among this universe of similar systems, the frequency of occurrence of a system with an actual WCET exceeding x is estimated at 1 in 10^{y}.

III. Conclusions

In this short paper, we have discussed the different meaning attached to pWCET distributions derived from Static Probabilistic Timing Analysis (SPTA) and Measurement-Based Probabilistic Timing Analysis (MBPTA) and their relation to aleatoric variability and epistemic uncertainty. These differences are crucial to the valid composition of these distributions in probabilistic schedulability analysis [15].

We conclude with a discussion of the interesting trade-off between epistemic uncertainty and aleatoric variability. Systems that exhibit *needle-in-a-haystack* problems, of rare very large execution times in the form of isolated outliers, present a serious challenge to any form of measurement-based timing analysis. Replacing the advanced hardware elements that lead to such isolated outliers with elements that exhibit randomised timing behaviour (for example random replacement caches) may smooth the timing behaviour making it more predictable (statistically), providing better management of epistemic uncertainty. Thus epistemic uncertainty may be reduced at a cost of increasing aleatoric variability. Since MBPTA can be applied to systems that exhibit both epistemic uncertainty and aleatoric variability, a number of interesting questions arise:

- Is there a benefit in trading epistemic uncertainty for aleatoric variability if the former cannot be completely eliminated?
- Is there any way to distinguish aleatoric variability from epistemic uncertainty in the results from MBPTA?
- How can we obtain a valid composition of pWCET distributions from different runs of different components?
- Is it beneficial to directly add aleatoric variability (random noise) to observations of a system that has epistemic uncertainty?
- Is a Static Probabilistic Timing Analysis necessary to show that there is no remaining epistemic uncertainty?
- How can we resolve the problem of representativity in MBPTA?

Acknowledgments

The ideas in this paper were first presented and discussed in an ad-hoc working group comprising Liliana Cucu-Grosjean, Adriana Gogonel, Iain Bate, Philipa Conway, Zoe Stephenson, Alan Burns and Robert Davis at the Dagstuhl Seminar on *Mixed Criticality on Multicore / Manycore Platforms* http://www.dagstuhl.de/17131. This work was funded in part by the Inria International Chair program and the ESPRC grant MCCps (EP/P003664/1). EPSRC Research Data Management: No new primary data was created during this study.

References

[1] S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Springer Real-Time Systems*, 51(1):77–123, 2015.

[2] S. Altmeyer and R. I. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 26:1–26:6, 2014.

[3] A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 89–96, 2000.

[4] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. Proartis: Probabilistically analyzable real-time systems. *ACM Transactions on Embedded Computing Systems*, 12(2s):94:1–94:26, May 2013.

[5] L. Cucu-Grosjean. Independence a misunderstood property of and for probabilistic real-time systems. *In Real-Time Systems: the past, the present and the future*, pages 29–37, 2013.

[6] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiones, and F. J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 91–101, July 2012.

[7] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 168–179, July 2013.

[8] J. L. Diaz, D. F. Garcia, K. Kim, C-G. Lee, L. Lo Bello, J. M. Lopez, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 289–300, 2002.

[9] J. L. Diaz, J. M. Lopez, M. Garcia, A. M. Campos, Kanghee Kim, and L. L. Bello. Pessimism in the stochastic analysis of real-time systems: concept and applications. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 197–207, Dec 2004.

[10] S. Edgar and A. Burns. Statistical analysis of wcet for scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 215–224, Dec 2001.

[11] J. Hansen, S. A. Hissam, and G. A. Moreno. Statistical-based WCET estimation and validation . In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 252, 2009.

[12] B. Lesage, D. Griffin, S. Altmeyer, and R. I. Davis. Static probabilistic timing analysis for multi-path programs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 361–372, Dec 2015.

[13] G. Lima and I. Bate. Valid application of evt in timing analysis by randomising execution time measurements. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.

[14] G. Lima, D. Dias, and E. Barros. Extreme value theory for estimating task execution time bounds: A careful look. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016.

[15] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 224–235, Dec 2013.

[16] L. Santinelli, F. Guet, and J. Morio. Revising measurement-based probabilistic timing analysis. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr 2017.

[17] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 21–30, 2014.

[18] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 241–248, June 2013.

Markov Chain Modeling of Probabilistic Real-Time Systems

Jasdeep Singh, Luca Santinelli, Guillaume Infantes, David Doose and Julien Brunel
ONERA Toulouse, name.surname@onera.fr

Introduction and Motivation

Real-time systems demand predictability in order to ensure safety. Applications, processors and resources have to be predictable in their behavior in order to have all the timing constraints guaranteed.

Real-time tasks are designed with Worst Case Execution Times (WCETs) which are deterministic values of the maximum amount of time the task can take to finish execution; however, in today's real-time systems a task execution resembles more to a random process than to a deterministic one. Frequent and unpredictable interferences between system elements, due to complex conservative modeling and lack of information, make task executions highly variable. Hence, probabilistic worst-case bounds are emerging as valid representations for the task execution behavior. For instance, the notion of WCET can be extended by the probabilistic WCET (pWCET) which is a probability distribution that upper-bounds all the possible task execution times. pWCETs allow for a flexible representation which account for multiple execution conditions, even those highly improbable which comes from pathological cases such as faults.

A real-time system in which at least one of the elemental parameters is a non-deterministic and probabilistic value is called probabilistic real-time system. Modeling time and schedulability analysis for them has the challenge of guaranteeing predictability from random variables representations.

Over the years, some approaches have been proposed for the schedulability analysis of probabilistic real-time systems, [5, 3, 7, 9]. They are all adaptation of deterministic schedulability techniques to the case of probabilistic task models with discrete distributions. The growing complexity of real-time systems (more interactions for more variable task behavior) and probabilistic models which tends to continuous distributions of pWCETs demands new techniques to design and validate real-time systems.

Formal methods are mathematically proven techniques which are applied to model complex systems. There exist also formal methods which make use of probabilities which compose probabilistic processes to model system dynamics. We believe that formal methods can cope with probabilistic real-time systems and offer *safe* and *accurate* representations of the system behavior. On top of formal models, it is possible to build probabilistic schedulability analysis approaches for verifying predictability and the respect of the timing constraints.

Probabilistic Real-Time Modeling

Probabilistic Modeling: The task pWCET is the worst-case distribution which upper-bounds any task execution time [4]. Given a continuous random variable $\overline{\mathcal{C}}$ defined in $[0, +\infty[$, the probability density function $f_{\overline{\mathcal{C}}}(x)$ of $\overline{\mathcal{C}}$ returns the probability that a value extracted from $\overline{\mathcal{C}}$ lies between c_1 and c_2 $[c_1, c_2]$: $Pr(c_1 \leq \overline{\mathcal{C}} \leq c_2) = \int_{c_1}^{c_2} f_{\overline{\mathcal{C}}}(x)dx$; the cumulative distribution function $F_{\overline{\mathcal{C}}}(x)$ of $\overline{\mathcal{C}}$ gives the cumulative probability for $\overline{\mathcal{C}} \leq x$. $F_{\overline{\mathcal{C}}}(x)$ is the integration of $f_{\overline{\mathcal{C}}}(x)$ in $[0, x]$: $F_{\overline{\mathcal{C}}}(x) = \int_0^x f_{\overline{\mathcal{C}}}(y)dy$; the inverse cumulative distribution function $\overline{F}_{\overline{\mathcal{C}}}(x)$ of $\overline{\mathcal{C}}$ gives the probability of exceeding x as the probability that $\overline{\mathcal{C}} > x$; $\overline{F}_{\overline{\mathcal{C}}}(x) = \int_x^{\infty} f_{\overline{\mathcal{C}}}(y)dy$.

Probabilistic timing analysis provides continuous pWCET estimates as combination of exponential distributions, [4, 2]. With exponential distributions, it is $f_{\overline{\mathcal{C}}}(x) = \lambda e^{-\lambda x}$ with λ the distribution rate parameter, $F_{\overline{\mathcal{C}}}(x) = 1 - e^{-\lambda x}$ and $\overline{F}_{\overline{\mathcal{C}}}(x) = e^{-\lambda x}$.

Markov Chain Modeling: A Markov Chain is a random process that satisfies the Markov property i.e. the memoryless property. It refers to the sequence of random variables with serial dependence only between adjacent periods/intervals and can describe systems that follow a chain of linked events, where what happens next depends only on the current state of the system, [8].

A Continuous-Time Markov Chain (CTMC) is a Markov chain with continuous time Markov processes. It is defined by a finite or countable state space S, a transition rate matrix Q with dimensions equal to that of the state space and initial probability distribution defined on the state space. In a CTMC model, state transitions follow exponential laws which can be viewed as time spent in a state as described by exponential distributions and state transitions are labeled with probability of being chosen. Within a CTMC model, there can exist deterministic state transitions i.e. only one ending state is possible and non-deterministic transitions i.e. more than one ending state are possible.

The time spent in a state $\mathcal{X}_t$ of a CTMC model is exponentially distributed by a rate Λ, while if there are m transitions going out of state $\mathcal{X}_t$, each with a rate λ_k, then: $\Lambda = \sum_{k=1}^{m} \lambda_k$. The probability $Pr(\lambda_a)$ of choosing a transition with rate λ_a is: $Pr(\lambda_a) = \frac{\lambda_a}{\sum_{k=1}^{m} \lambda_k}, a \leq m$.

The system modeled as a CTMC is explored using model checking, [6]; information like "probability of being in a state $\mathcal{X}$ at time t" are extracted and formally proved. Steady state probabilities and time reversibility are interesting properties of CTMC models; they make it possible to model stable and correct system behavior.

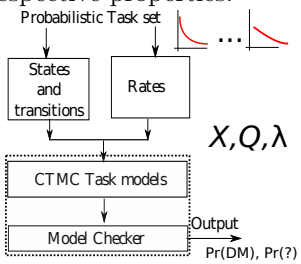
The CTMC can be applied for modeling probabilistic real-time systems, in particular tasks interactions and executions. The exponential pWCET estimates are perfectly coped with CTMCs, while the exponential internal behavior needs to be safely/pessimistically addressed. The steady state property is very handy for validating real-time systems with continuous input distributions and continuous system behaviors.

Job Modeling: In a periodic task model, a task τ composes of an infinite sequence of jobs J, such that as the task is dispatched, a job is released and is ready for execution. The job executes at the state E and after a certain amount of time, it can either finish execution and move to state F or get preempted and move to state P_1, where it waits for executing back. In P_1 the job can either

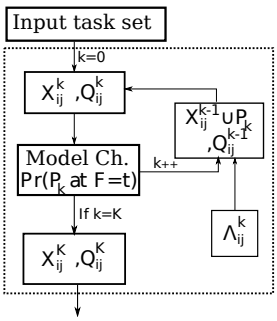
finish or get preempted again and move to state P_2, and so on until P_K where K is the maximum number of preemptions the job can have. The set of states $M = \{E, F, (P_k)_{k \leq K}\}$ describes the possible states for a job. E is job executing, F is when the job finished execution and $(P_k)_{k \leq K}$ is job executing after the k^{th} preemption from other jobs; $(P_k)_{k \leq K}$ includes in itself any delay/interference caused by the preempting jobs.

For each job J_{ij} it is possible to build a CTMC model given the input task and the representations of the tasks/jobs interactions such as the scheduling policy applied. The job CTMC model S_{ij} is $S_{ij} = \{\mathcal{X}_{ij}, Q_{ij}\}$, with $\mathcal{X}_{ij}$ the set of states for J_{ij}, $\mathcal{X}_{ij} \in M$ and Q_{ij} is the transition matrix describing the transitions between job execution states. The model building has to take into account for the job interactions e.g., preemptions and precedence constraints/priorities, and the job pWCETs distributions. The task τ_i representation composes of all the CTMC job models J_{ij} of τ_i, $\{\{\mathcal{X}_i\}, \{Q_i\}\}$. With the hyperperiod valid, the jobs to account for are those within the hyperperiod. A probabilistic real-time system is represented by grouping together tasks and their respective properties.

From the CTMC models of real-time systems, real-time properties can be extracted. There would exist the probability to finish execution by time t $Pr(F \leq t)$, probability to get preempted by time t $Pr(P_1 \leq t) + Pr(P_2 \leq t) + \ldots + Pr(P_K \leq t)$, and the deadline miss ratio $Pr(DM)$. All those properties can be validate at the steady states, defining the correct behavior of the system at time equals infinite. Figure 1(a) describes how to build and validate CTMC models from input pWCETs. Figure 1(b) provides an insight on the iterative process for building job CTMC models.



(a) CTMC modeling



(b) Job CTMC iterative model

Figure 1: Constructing CTMC models.

Open Problems

There are various open problems to the formal method approach for probabilistic real-time systems. *Formal and pessimistic/safe by construction*: CTMC job models have to be formally built from task/job representations (pWCET, periods, deadlines) and the scheduling policy applied. It means that all the interference/preemptions have to be taken into account and every operation applied e.g., exponential upper bounding, interference computation, cannot introduce any optimism to the task/job behavior. Pessimistic operators for CTMC models are currently under definition. We are also considering the CTMC model built iteratively (and iteratively model checked), where at each step the number of possible interactions between interfering jobs is increased. This is for allowing a formal model construction and incremental formal validation. As a remark, in [5] Markov chain modeling has been applied to study only the load of the system; with CTMC we aim at probabilistically modeling the scheduling of the tasks and all of their interactions at runtime.

System properties: real-time properties can be extracted from CTMC models e.g. deadline miss ratio, and proved to be corrected at the steady states; other system properties have yet to be extracted from the models through model checking. For instance, the precedence constraints imposed by the scheduler can be formally proved with CTMC models, which would mean proving the scheduler. We foresee the possibility of validating system functional behavior e.g., precedence constraints and the scheduling policies from the models and model checking. Furthermore, time reversibility property of Markov chain has to potential to model self correcting probabilistic real-time systems. We plan to explore it in the near future. With respect to precedence constraints, we will prove that hybrid continuous time and discrete time Markov Chains models would allow approaching such cases by embedding dependence between task executions.

Enhanced/more formal models: the CTMC model is not the only possible model for probabilistic real-time system. Former works have approached stochastic petri nets [1]. There is the need to continue such investigation in order to be able to integrate deterministic and probabilistic task sets, discrete and continuous probabilistic models, etc.. For instance, probabilistic time automata will be studied with WCET thresholds $\langle C, p \rangle$ in order to avoid state explosion; the WCET threshold $\langle C, p \rangle$ is such that: C is the WCET value extracted from $\overline{\mathcal{C}}$ such that $\overline{F}_{\overline{\mathcal{C}}}(C) = p$.

References

[1] L. Carnevali, A. Melani, L. Santinelli, and G. Lipari. Probabilistic deadline miss analysis of real-time systems using regenerative transient analysis. In *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*, page 299, 2014.

[2] F. J. Cazorla, T. Vardanega, E. Quinones, and J. Abella. Upper-bounding program execution time with extreme value theory. In *WCET*, 2013.

[3] L. Cucu and E. Tovar. A framework for response time analysis of fixed-priority tasks with stochastic inter-arrival times. *ACM SIGBED Review*, 3(1), 2006.

[4] 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 *Euromicro Conf. on Real-Time Systems (ECRTS)*. IEEE, 2012.

[5] J. Díaz, D. Garcia, K. Kim, C. Lee, L. Bello, L. J.M., and O. Mirabella. Stochastic analysis of periodic real-time systems. In *23rd of the IEEE Real-Time Systems Symposium (RTSS02)*, pages 289–300, 2002.

[6] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, LNCS, pages 585–591. Springer, 2011.

[7] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *IEEE Real-Time Systems Symposium*, 2013.

[8] J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[9] L. Santinelli and L. Cucu-Grosjean. A probabilistic calculus for probabilistic real-time systems. *ACM Trans. Embedded Comput. Syst.*, 14(3), 2015.

A Mixed Criticality Approach for Industrial Smart Energy Management and Demand Response

Alemayehu Addisu
METRON/UPEM, France
alemayehu.addisu@metronlab.com

Laurent George, Hakim Badis
UPEM/LIGM-ESIEE, France
{laurent.george,hakim.badis}@u-pem.fr

Pierre Courbin
METRON, France
pierre.courbin@metronlab.com

I. Introduction

In a power system, demand response (DR) is regarded as an effective approach to increase electricity grid performance and consumer benefits [1]. With DR, a factory, for example, is requested by a legacy energy provider to stop or reduce its power consumption and can get payed for not consuming. The legacy energy provider can then smooth peaks of energy consumption at country level with DR instead of buying energy at the highest price on spot markets. The deadline to accept or decline a DR ranges from 10 minutes to 1 hour. DR programs throttle energy demands of different loads such as industrial, commercial and residential for adjusting demands to the available production. Control of electricity consumption can be in the form of price- or quantity-based DR. In the price-based DR programs such as Time-of-Use (TOU) and Real-Time Pricing (RTP), informing customers of an increase in electricity price will make an incentive for them to reduce their energy consumption. In the quantity-based DR programs, it limits the loads of customers by imposing upper bound on amount of electricity consumption.

Being large electricity consumers, manufacturing sectors face challenges of increased energy cost and a need to reduce their carbon footprint. These challenges encourage decision-makers to look for viable solutions. Demand Side Management (DSM) could be one solution to alleviate these challenges by adjusting industrial processes to DR programs. According to a study performed in [2], 20 top USA industries have at least 12GW of load flexibility available for DR. To achieve this huge potential, identifying candidate industrial processes is a crucial step. In multi-product manufacturing systems (e.g., manufacturing of semiconductor chips and other electronic products), there can be many product lines with different importance levels. Some of the lines could be highly critical for overall manufacturing (we name them HI-crit lines) and others could be less important (we name them LO-crit lines). When energy is available (or electricity price is low), we would like to operate at full capacity to produce items at a maximum production rate. However, during DR periods, we need to reschedule the processes (by at least keeping the Hi-crits) to minimize cost without reducing production rate too much. Hence, scheduling in such manufacturing types give rise to a mixed criticality (MC) problem.

In MC systems, multiple functionalities are implemented upon a single shared computing platform. Typically, all these different functionalities do not possess the same level of criticality to the overall system performance. This concept of mixed criticalities gave rise to a mixed-criticality scheduling problem. It was initially introduced by Vestal [3] in 2007. In MC scheduling, more pessimistic assumptions are required in order to validate the correctness of the highly critical functionalities than the others with lower criticality levels. Reviews of MC systems are provided in [4] and [5]. In this paper, we raise a scheduling problem in a multi-product manufacturing system. We consider as an illustrative example one of the manufacturing types listed in [6], specifically production serial lines with multi-product. In this types of manufacturing systems, a conveyor system synchronizes the stations/machines, i.e., it literally sets a *hard deadline*. Every station should finish processing before this deadline. The HI-crit and LO-crit lines use different conveyors. The objective is to construct a *scheduling table* which achieve the highest production rate possible. Under normal condition, all HI-crit and LO-crit lines can work. However, we need to construct a new scheduling table during DR periods. In the next section, we formally define the problem.

II. Problem Description

In this section, we describe the scheduling problem in a flow line (or serial line) manufacturing system shown in Fig. 1. Let us assume that the system contains two product lines for two product types. Stations/machines $M_{11}, M_{12}, M_{13}, M_{31}$, and M_{32}

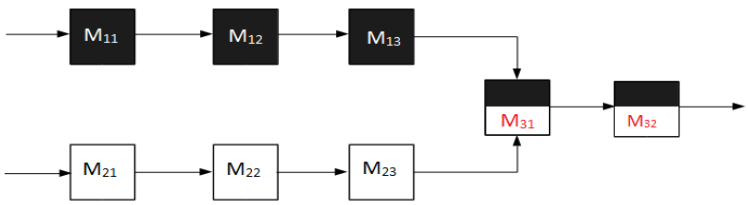


Fig. 1. A serial production line with two product types

process the first product type. We may say that it is a HI-crit product. The second product type (or LO-crit product) goes under stations $M_{21}, M_{22}, M_{23}, M_{31}$, and M_{32} where stations M_{31} and M_{32} process both product types. From Fig. 1, we can see that there are three conveyors for synchronization among the stations. That is, conveyor 1 moves the HI-crit products between stations M_{11} to M_{13}, conveyor 2 moves LO-crit products between stations M_{21} to M_{23}, and conveyor 3 moves both HI- and LO-crit products between stations M_{31} to M_{32}. Each conveyor is blocked until a station having the maximum processing time finishes. Hence, respective deadlines for the stations correspond to these times.

TABLE I
SCHEDULING TABLE UNDER NORMAL AND DR CONDITIONS

Time	M_{11}	M_{12}	M_{13}	M_{21}	M_{22}	M_{23}	M_{31}^{HI}	M_{31}^{LO}	M_{32}^{HI}	M_{32}^{LO}
t_1	1	1	1	1	1	1	1	0	1	0
t_2	1	1	1	1	1	1	0	1	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t_i	1	1	1	0	0	0	0	0	0	0
t_{i+1}	0	0	0	1	1	1	0	0	0	0
t_{i+2}	1	0	0	0	0	0	1	0	1	0
t_{i+3}	0	1	1	0	0	0	0	1	0	0
t_{i+4}	0	0	0	1	0	0	1	0	0	1
t_{i+5}	0	0	0	0	1	1	0	0	1	0

A scheduling table for the serial production line (refer to Fig. 1) is given in Table I. This scheduling table can be constructed according to a given available power. In the table, the "1"s and "0"s represent ON/OFF states of the stations, respectively. Assume that t_1 = 0 and t_2 - $t_1 \geq \max(C_{M_{ij}})$, where $C_{M_{ij}}$ is the processing time of station M_{ij}. We assume the same processing times for LO- and HI-crit products for the sake of simplicity. This is not necessarily the case in the general case. Under normal conditions, the production rates of HI- and LO-crit products are both 1/2 products per time unit. This is labeled as "case 1" in Fig. 2. However, when DR starts at t_i (case 2 in Fig. 2), the available power is not enough to schedule all the stations at the same time. A simple schedule can be to turn ON only stations on the HI-crit line and turning OFF the others at time t_i. In case 2, production rate of HI-crit products is 2/6 and that of LO-crits is 1/6. In most cases, finding a new schedule on-line may not be simple in manufacturing systems. Hence, it necessitates a new schedule to respect the upper bound on power consumption while maximizing the production rate of HI-crit products . The new scheduling table should also satisfy some fixed limit on production rate of HI-crit products. We may not produce LO-crit products at all if the DR is too much (case 3 in Fig. 2). Another issue is how does the scheduling table evolve if there are local energy resources such as solar, battery, etc? Can we propose an on-line algorithm to adapt production, not necessarily based on a scheduling table, where available power thresholds are limited?

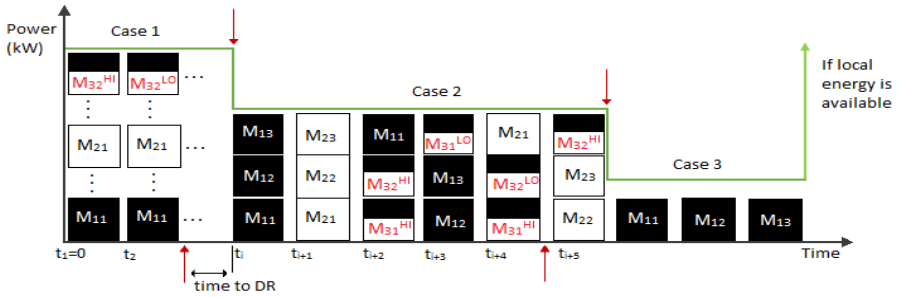


Fig. 2. Available power constraint levels (upper bound due to DR). Height of a box is power consumption and its width is processing time. The sizes of the boxes can be different based on power consumptions and processing times.

REFERENCES

[1] Q. Zhang and I. Grossmann, *Planning and scheduling for industrial demand side management: advances and challenges*, Alternative Energy Sources and Technologies, Springer, pp. 383–324, 2016.

[2] M. Starke, N. Alkadi, and O. Ma, *Assessment of Industrial Load for Demand Response across US Regions of the Western Interconnect*, Oak Ridge National Laboratory, 2013.

[3] S. Vestal, *Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance*, Real-Time Systems Symposium, RTSS 2007. 28th IEEE International, pp. 239–243, 2007.

[4] A. Burns, and R. Davis, *Mixed criticality systems - a review*, Department of Computer Science, University of York, Tech. Rep, 2013.

[5] S. Baruah, L. Cucu-Grosjean, R. Davis, and C. Maiza, *Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121)*, Dagstuhl Reports, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, VOL. 5, NO. 3, 2015.

[6] J. Li, D. E. Blumenfeld, N. Huang, and J. M. Alden, *Throughput analysis of production systems: recent advances and future topics*, International Journal of Production Research, Taylor & Francis, VOL. 47, pp. 3823–3851, 2009.

Estimating Worst-Case Bounds for Open CPS Runtimes with Genetic Algorithms

Oliver Horst
fortiss GmbH
Munich, Germany
Email: horst@fortiss.org

Uwe Baumgarten
Technical University of Munich,
Department of Informatics, Germany
Email: baumgaru@tum.de

Christian Prehofer
fortiss GmbH
Munich, Germany
Email: prehofer@fortiss.org

"A key requirement for cyber-physical systems is that they are open. They should be readily extensible and at the same time future proof, so that they can meet requirements that have not even been thought of yet.", Jörg P. Müller, TU Clausthal

Hence, open cyber-physical systems (CPS) have to cope with unknown deployment configurations of almost arbitrary applications. Furthermore, CPSs are designed to interact with the physical world. Thus, their temporal behavior may be critical for correct functional behavior. Accordingly, worst-case bounds (WCBs), e.g., the application's execution time and the utilized resources, need to be determined to guarantee correct and dependable execution.

Analyzing the worst-case execution time (WCET) is in general done at the system level, with knowledge of all software components. This is not feasible for open CPSs, as the overall system configuration is not known at design time. Accordingly, timing compositional platforms [1] and analysis [2] where proposed. These approaches aim at providing independent analysis of the involved software components and a later composition of the overall WCB. However, these analyses still require the WCBs of all involved software components to calculate the actual WCET of any deployed application. This is impracticable for developers of open CPS applications. A time compositional runtime environment is a key requirement to enable truly individual WCET analysis for open CPSs [1, 2]. For the development and deployment of such runtimes, it is important to determine the worst-case response time (WCRT) of every possible interaction of applications with the runtime itself. As we cannot make any safe assumptions about the deployed applications, every possible deployment scheme of all valid applications needs to be considered to determine the WCBs of a runtime API call. This is clearly a huge and complex search problem. Contemporary multi-core platforms add a further dimension to this challenge. Determining the WCET for applications on these platforms is still a research challenge, even for known system configurations, not even taking into account the aspect of open platforms [3].

I. Genetic Algorithm based Evaluation

To provide developers and users of such runtimes with an estimation of the WCBs of their API calls, we propose the use of a genetic algorithm (GA) to evolve execution scenarios that provoke the worst-case time/resources/etc. usage of a specific API call. Thus, we turn the search problem into an optimization problem. This concept was proposed earlier for estimating WCETs [4]. Each execution scenario describes a certain deployment configuration and environmental circumstances for its execution. Scenarios are represented by the following triple: (*i*) a task set, (*ii*) an interrupt set, and (*iii*) a message set. The task set (*i*) describes the deployed tasks. The interrupt set (*ii*) describes interrupts that occur during the execution scenario. The message set (*iii*) describes task interactions with the underlying runtime. We assume that these interactions take place by exchanging messages, which could also represent an API call. Fig. 2 illustrates an example scenario triple in the form of timelines. We are using the following notation for event specifiers in the figure. The first letter marks the type of the event: T for task, I for interrupt, or M for message. The global ID of the event follows in square brackets, and the processing time of the event comes after the colon. If the event is a task, the type of task will be denoted after a comma (S for sporadic and P for periodic). Fig. 2a illustrates an example task deployment for two cores, which also specifies the occurring interrupts in the described scenario. Fig. 2b shows the interaction plan of a sporadic task. The bar in Fig. 2b represents the execution time, as specified in Fig. 2a. The two message events indicate what kinds of messages are released by the task.

To allow a GA to generate such scenarios, it has to be formulated in form of a gene-model, i.e., bitstring. For the evolution process within the GA, moreover, a fitness function is needed that concludes a WCB from such a gene-model. Our fitness function is bisected into a simulation part and an assessment part. The simulation takes a scenario in its bitstring representation, transforms it into an input understood by the runtime under assessment, and let the runtime determine a system-wide schedule for the scenario, which is then assessed regarding the WCB in question. The assessment results in turn steer the next evolution round. The overall concept is shown in Fig. 1. We have split the simulation into four parts: the communication middleware, the task scheduler, a resource arbiter, and a hardware model. The split was made to support the direct evaluation of individual contributions to any of the four fields. We expect the middleware to be responsible for providing the overall interface to the runtime environment. It utilizes a task scheduler for executing communication and processing tasks, and coordinates accesses to peripherals via a resource arbiter. Finally, the calculated schedules are simulated, i.e., evaluated, on the hardware model, to obtain the actual schedule which ideally respects all relevant hardware interferences (see also Section II-A).

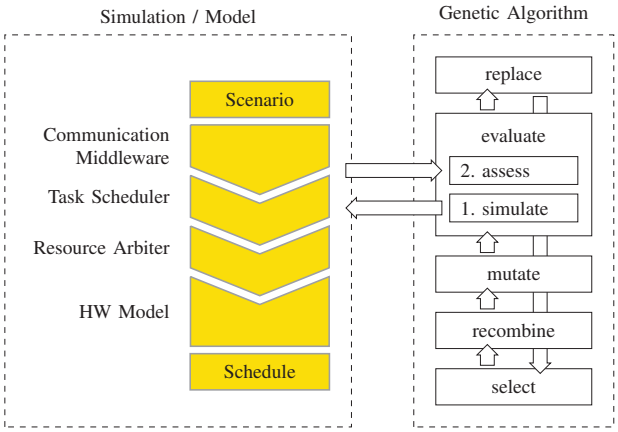


Figure 1. Conceptual view on the evaluation framework, showing the interworking between the genetic algorithm and the simulation/model.

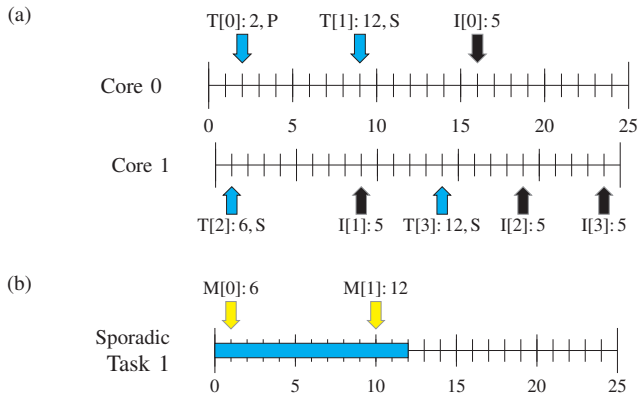


Figure 2. Illustration of the scenario triple, in form of the task and interrupt model (a) and the message model (b).

II. Open Questions

A. Level of Detail of the Hardware Model

Our first question concerns the level of detail of the hardware model. The model should cover all known relevant interference effects from today's multi-core platforms, but be at the same time it should be as simple as possible without introducing an additional uncertainty to the overall approach. We see the following options: (*i*) very fine grained models and simulations based on VDHL or SystemC code like ARM Cycle Models, (*ii*) platform simulators like OVP or gem5, or (*iii*) specifically tailored simulators that try to focus on the relevant parts of the model like ERAsim. As indicated by [5] there are effects on various layers that might affect the overall timing behavior of current multi-core processors, it is not yet clear where we can simplify the model and how, or which model layers are really relevant for such a top-down analysis as we are proposing it.

B. Quantification of the Worst-Case Bound Quality

GAs are heuristic optimization algorithms, hence, our evaluation approach can only estimate WCBs and cannot give guarantees that it will actually find the actual worst-case. Thus, we face the question whether there is some way to determine a time constraint in which the GA leads to a result of a guaranteed quality in the sense of a bounded maximal under/over estimation of the determined WCB. This requires an assessment strategy for the GA result deviation from the actual WCB. [6] for instance showed that the estimated WCET obtained with a GA lays in the range of $\pm 10\%$ of the real WCET for some known problems. The same evaluation concept could be used for our approach. Another possibility we see in utilizing well known benchmarking functions for GAs to measure our accuracy. Finding WCBs for interactions with an open CPS runtime is mathematically comparable to a function with (*i*) a multi-dimensional input, (*ii*) many local maxima, (*iii*) multiple global maxima, (*iv*) plateaus and valleys, and (*v*) cross linked dimensions. This is in line with [6] regarding the execution time function of bubblesort. We consider the following functions from the Black-Box Optimization Benchmarking (BBOB) workshop as candidates for an evaluation of the accuracy: f105- Rosenbruck, f113- Step ellipsoid, f119- Different Powers, f122- Schaffer's F7, f126- Composite Griewank-Rosenbrock, and f128- Gallagher's Gaussian Peaks 101-me. See [7] for a mathematic definition of these functions.

Acknowledgment

The research leading to these results was funded by the European Union (EU) through the project TAPPS (Trusted Apps for open CPSs).

References

[1] A. Baldovin, E. Mezzetti, and T. Vardanega, *Towards a Time-Composable Operating System*. Springer, 2013, pp. 143–160.
[2] S. Hahn, J. Reineke, and R. Wilhelm, "Towards compositionality in execution time analysis: Definition and challenges," *ACM SIGBED Review*, vol. 12, no. 1, 2015.
[3] M. Jacobs, S. Hahn, and S. Hack, "A framework for the derivation of WCET analyses for multi-core processors," in *28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.
[4] I. Bate and U. Khan, "WCET analysis of modern processors using multi-criteria optimisation," *Empirical SW Eng.*, vol. 16, no. 1, 2011.
[5] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multicore in real-time systems – temporal isolation challenges due to shared resources," CISTER Research Unit, ISEP-IPP, techreport 140302, 2014.
[6] J. Wegener and F. Mueller, "A comparison of static analysis and evolutionary testing for the verification of timing constraints," *Real-Time Systems*, vol. 21, no. 3, 2001.
[7] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-Parameter Black-Box Optimization Benchmarking 2009: Noisy Functions Definitions," INRIA, techreport RR-6869, 2009.

On the Existence of a Cyclic Schedule for Non-Preemptive Periodic Tasks with Release Offset

Mitra Nasri
Max Planck Institute for Software Systems, Germany.
mitra@mpi-sws.org

Emmanuel Grolleau
ENSMA Chasseneuil, France.
grolleau@ensma.fr

I. Introduction

Non-preemptive scheduling reduces both runtime and design time overhead since it avoids context switches, preserves cache affinity, and simplifies the use of shared resources. As a result, it increases the accuracy of worst-case execution time (WCET) estimation and improves system predictability. Many safety critical systems use table driven scheduling in which tasks are dispatched according to a schedule stored in a table. This table contains a cyclic schedule that needs to be repeated during the lifetime of the system. Since the table must be kept in memory, a smaller table is preferable particularly for embedded systems with limited memory. For example, the Atmel UC3A0512 microcontroller, which is used in mission critical space applications [1], has 64 KiB of internal SRAM, 512 KiB of internal flash memory, and is clocked at 12 MHz. Similarly, an Arm Cortex-M3 has 4 KiB RAM, 16 KiB flash, and clock speed 24 MHz. With such limited resources, non-preemptive execution becomes the natural way of executing real-time tasks in these systems [2].

In a non-preemptive scheduling table, the number of entries is analogous with the number of instances of the tasks (a.k.a jobs) in a cyclic schedule. It is known that in a synchronous periodic task set with constrained deadlines, the schedule becomes cyclic after H units of time where H is the least-common-multiple (LCM) of periods [3] for any feasible schedule. However, many systems use release offsets for their tasks in order to enforce particular properties such as avoiding the critical instant, or enforcing precedence constraints. In that case, a cyclic schedule may not be found in the first hyperperiod, which then increases the size of scheduling table. In our work, we are interested in the length of a cyclic schedule since many embedded systems have limited amount of memory to store the table.

Choquet-Geniet et al. [4] have shown that any preemptive periodic task set with independent tasks and constrained deadlines that is feasible in a work-conserving schedule, satisfies the following conditions (we call it after the authors' names *Choquet Grolleau Conditions* (CGC)):

- **(i)** it becomes cyclic after time $t^{start} \in [0, O^{\max} + H]$ where $O^{\max}$ is the largest release offset,
- **(ii)** the length of cycle is H,
- **(iii)** the cycle is the first window of length H in the schedule having a slack exactly equal to $(1 - U) \cdot H$, where U is the task set utilization,
- **(iv)** there is no deadline miss from time 0 to the end of the cycle.
- **(v)** every job appearing in the cycle starts and ends in the cycle.

Goossens et al. [5] have extended [4] and derived an upper bound on the length of the cycle for any feasible schedule (work-conserving or not, uni or multiprocessor). However, this upper bound is a function of task's periods, deadlines, and release offsets and may become exponential with respect to H.

In this work, we study the existence of a cyclic schedule for non-preemptive tasks when they are scheduled by non-work-conserving scheduling algorithms. In the remainder of the paper we consider a set of non-preemptive periodic tasks denoted by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task τ_i is identified by its execution time C_i, its period T_i, its deadline $D_i \leq T_i$, and its release offset O_i. We denote the least-common-multiple (LCM) of the periods by H and utilization of the task set by $U = \sum_{i=1}^{n} C_i/T_i$. Moreover, it is assumed that the system has a dispatcher to dispatch the jobs one after another according to the start times stored in the table, hence, the schedulability of the task set will not be affected by runtime release jitters or execution time variations.

In Sec. II, by an example we show that there exists online non-work-conserving scheduling algorithms that create a cycle with length $2H$ (i.e., they violate condition (ii) of CGC). We also show that it is possible to modify our example schedule such that it satisfies condition (ii) of CGC although it will no longer be consistent with the scheduling algorithm. In Sec. III, we formally introduce the open problems.

II. Motivating Examples

Precautious-RM is one of the recently introduced online non-work-conserving scheduling algorithms for non-preemptive periodic tasks [6]. It priorities the tasks according to the rate monotonic (RM) policy, however, before scheduling a task such as τ_i at time t, Precautious-RM verifies the following condition: $t + C_i \leq r_1^{next} + D_1 - C_1$, where r_1^{next} is the expected release

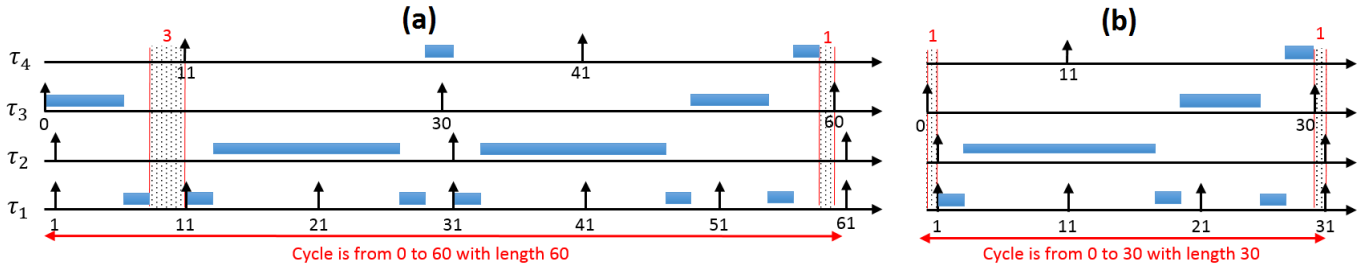


Fig. 1. A task set with 4 tasks scheduled by (a) Precautious-RM [6] and (b) an optimal schedule. In this task set $C_1 = 2$, $C_2 = 14$, $C_3 = 6$, $C_4 = 2$, $T_1 = 10$, $T_2 = T_3 = T_4 = 30$, $O_1 = O_2 = O_4 = 1$, and $O_3 = 0$. Here, H is 30 and $U = \frac{28}{30}$.

time of the next job of τ_1. If the condition holds, it schedules τ_i, otherwise it schedules an idle-interval from t to r_1^{next}. In other words, it schedules a low priority task only if it will not cause a deadline miss for the next job of τ_1. If Precautious-RM is used to create a non-preemptive schedule for the following task set, the resulting schedule will have a cycle with length $2H$ as it is shown in Fig. 1.

As it can be seen in Fig. 1-(a), Precautious-RM has inserted an idle-time from time 8 to 11 because if τ_2 was executed at 8, it would have caused a deadline miss for the second job of τ_1. Even though the length of cycle is $2H$ in Fig. 1-(a), the total amount of slack is 4 which is equal to $(1 - U) \cdot 2H$. In Fig. 1-(b), an optimal schedule has been presented that schedules the task set in a cycle with length H. This schedule satisfies all CGC.

III. Open Problems

The first open problem is about the existence of a task set for which none of the feasible schedules satisfies CGC. The existence of such task set implies that memory required for storing an offline table of a set of non-preemptive tasks with release offsets might become unbounded or unreasonably large. Moreover, Since the problem of scheduling a set of non-preemptive periodic tasks is known to be strongly NP-Hard (see Jeffay et al. [7]), not knowing the existence of a cycle with a limited length will only make the situation worse. It means that in order to find a schedule, a large number of jobs must be considered (if a cycle exists).

Problem 1. Find a feasible task set τ for which there does not exist a cyclic schedule or the length of cycle is larger than H.

If there exists a task set that matches with Problem 1, then the next question is how to find a cyclic schedule such that the number of cycles is minimized. Otherwise, if there is no such task set, then we will be interested to find the CGC-compatible schedule. Next we define these two follow-up problems.

Problem 2. Given a feasible non-preemptive task set τ, find a schedule S that has the smallest cycle length and satisfies condition (iv) of CGC.

Problem 3. Given a feasible non-preemptive task set τ, find a schedule S that satisfies CGC.

It is worth noting that we still do not know whether an answer to Problem 2 satisfies condition (i) of CGC or not. This answer will most probably satisfy condition (iii) because that is the notion of reaching to an equilibrium state.

Acknowledgment

This work is supported by a fellowship from Alexander von Humboldt Foundation. The authors would like to thank Schloss Dagstuhl for seminar number 17131 that helped this work to happen.

References

[1] J. F. Ruiz, "GNAT Pro for On-board Mission-Critical Space Applications," in *Ada-Europe International Conference on Reliable Software Technologies*, 2005, pp. 248–259.

[2] B. Nasri, Mitra andBrandenburg, "Offline Equivalence: A Non-Preemptive Scheduling Technique for Resource-Constrained Embedded Real-Time Systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017.

[3] J. Y.-T. Leung and M. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information processing letters*, vol. 11, no. 3, pp. 115–118, 1980.

[4] A. Choquet-Geniet and E. Grolleau, "Minimal schedulability interval for real-time systems of periodic tasks with offsets," *Theoretical computer science*, vol. 310, no. 1-3, pp. 117–134, 2004.

[5] J. Goossens, E. Grolleau, and L. Cucu-Grosjean, "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms," *Real-Time Systems*, vol. 52, no. 6, pp. 808–832, 2016.

[6] M. Nasri, G. Nelissen, and G. Fohler, "A new approach for limited preemptive scheduling in systems with preemption overhead," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 23–35.

[7] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 1991, pp. 129–139.

Fixed Priority Scheduling of xy-tasks

Maksim Kavalerov
Department of Automation and Telemechanics
Perm National Research Polytechnic University, Russia
Email: mkavalerov@gmail.com

I. Introduction

The paper addresses the problem of fixed priority scheduling with timing constraints imposed on inter-job relationships of a task. The model of xy-tasks is proposed for this purpose. Fig. 1 represents the location of this model among some well-known task models, such as: Liu and Layland [1], sporadic [2], multiframe [3], generalized multiframe (GMF) [4], non-cyclic GMF [5], recurring RT (RRT) [6], non-cyclic RRT [7], Digraph (DRT) [8], k-EDRT [8], EDRT [8]. Fig. 1 is taken from [8], slightly modified, and the location of xy-task model is added to the picture.

In the most general formulation, xy-task model presumes that xy-tasks have generalized timing contstraints, and this makes schedulability analysis of xy-tasks intractable in general. The notion of interval timing constraints is proposed in [9] to facilitate schedulability analysis and attribute assignment. Thus, xy-tasks with interval timing constraints can be considered as a specialization of xy-task model. Several open scheduling problems related to xy-task model and its specialization are introduced below.

II. xy-task Model

All tasks are independent, each task τ_i has its own timing constraint. Tasks are scheduled on a single processor according to *priority preemptive* policy. Any job $\tau_{i,j}$ is released and ready for execution at its *release time* $r_{i,j}$. The *start time* and the *finish time* of job $\tau_{i,j}$ are denoted by $s_{i,j}$ and $f_{i,j}$ respectively. We assume that the execution interval $[r_{i,j}, f_{i,j}]$ of each job $\tau_{i,j}$ contains two *special instants* denoted by $x_{i,j}$ and $y_{i,j}$. Sampling and actuation instants of job $\tau_{i,j}$, related to *control* task τ_i, can be represented as $x_{i,j}$ and $y_{i,j}$ which may not be equal to $r_{i,j}$, $s_{i,j}$, or $f_{i,j}$ due to execution of the job code before sampling and after actuation. Thus, consider a task τ_i and its execution with interruptions. There are 5 instants during execution of each job $\tau_{i,j}$: $r_{i,j}$, $s_{i,j}$, $x_{i,j}$, $y_{i,j}$, $f_{i,j}$. And there are 10 interval lengths: $s_{i,j} - r_{i,j}$, $x_{i,j} - r_{i,j}$, etc. We call them *job execution parameters*. We denote these lengths by $rs_{i,j}$, $rx_{i,j}$, $ry_{i,j}$, $rf_{i,j}$, $sx_{i,j}$, $sy_{i,j}$, $sf_{i,j}$, $xy_{i,j}$, $xf_{i,j}$, $yf_{i,j}$, see Fig. 2.

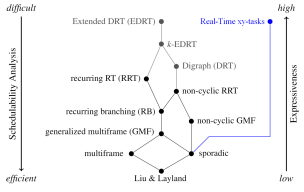


Fig. 1. A hierarchy of task models taken from [8] along with the proposed xy-tasks model

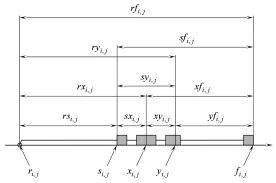


Fig. 2. Job execution parameters

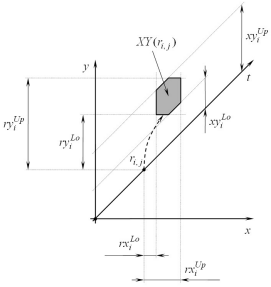


Fig. 3. The graphical representation of $XY(r_{i,j})$

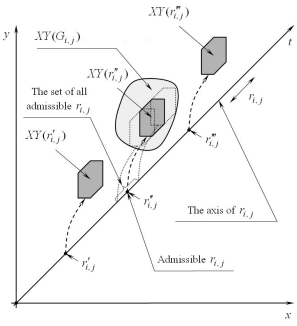


Fig. 4. Admissible $r_{i,j}$

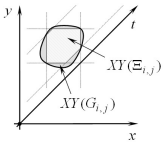


Fig. 5. Ξ_i can be considered as an approximation of G_i

Each job execution parameter can be estimated with lower and upper bounds, e.g., using best-case and worst-case response time analysis. We denote these estimates by rs_i^{Lo}, rs_i^{Up}, etc. And $rs_{i,j} \in [rs_i^{Lo}, rs_i^{Up}]$ for $\forall j \geq 1$, etc. These estimates can be obtained by using worst-case and best-case estimation techniques, for example, such as those presented in [10] and [11]. An approach how to calculate these estimates for job execution parameters is proposed in [12]. Also, for each task τ_i some external information is available to the scheduler as a set of vectors $\{\xi_{i,j}\}$, where each vector $\xi_{i,j}$ is available no later than $f_{i,j}$ and may contain information that possibly affects the timing constraint of task τ_i and its release pattern. *Quality function* over first j jobs in its most general form is represented as a function $Y_{i,j}(x_{i,1}, y_{i,1}, \xi_{i,1}, x_{i,2}, y_{i,2}, \xi_{i,2}, \ldots, x_{i,j}, y_{i,j})$. A simple example is $Y_{i,j} = \max_{\forall m \leq j}(-\frac{1}{w}\sum_{k=m-w+1}^{m}(\alpha(x_{i,k} - x_{i,k-1}) + \beta(y_{i,k} - x_{i,k})))$, where α and β are constants, w is the number of consecutive jobs which are subject for averaging. For this example, it is assumed that $x_{i,k} = r_{i,k}, \forall k \geq 1$, and $x_{i,0}$ is predefined. This quality function is quite similar to the cost functions for a control task presented in [13]. If *guaranteed* quality is required, as in the case of *hard* real-time tasks, then the objective is to *guarantee* that $Y_{i,k} \geq Y_i^{Lo}$ (if task τ_i has only k jobs) or $\lim_{j\to\infty} Y_{i,j} \geq Y_i^{Lo}$ for a predefined Y_i^{Lo}. Many timing constraints of control tasks can be represented in the same way.

III. Problem Statement and Known Results

We limit our scope to *fixed priority preemptive* scheduling. One can affect $x_{i,j}$, $y_{i,j}$ of a job by choosing an appropriate $r_{i,j}$. Let $XY(r_{i,j})$ be the set of all pairs $(x_{i,j}, y_{i,j})$ for a given $r_{i,j}$, such that the following conditions hold: $x_{i,j} \in [r_{i,j}+rx_i^{Lo}, r_{i,j}+rx_i^{Up}]$, $y_{i,j} \in [r_{i,j}+ry_i^{Lo}, r_{i,j}+ry_i^{Up}]$, $xy_{i,j} \in [xy_i^{Lo}, xy_i^{Up}]$. Fig. 3 shows the graphical representation of $XY(r_{i,j})$. Notice that each point in the shaded area represents a pair $(x_{i,j}, y_{i,j})$ that satisfies the above-mentioned conditions. The diagonal axis is scaled in such a way that the a point, projected onto axis x or y, gives the value $r_{i,j}$. Let $G_{i,j}$ denote the constraint $Y_{i,j} \geq Y_i^{Lo}$. Let $XY(G_{i,j})$ denote a set of all pairs $(x_{i,j}, y_{i,j})$ such that $G_{i,j}$ holds. $G_{i,j}$ *is guaranteed to be satisfied* for a given $r_{i,j}$, if $XY(r_{i,j}) \in XY(G_{i,j})$. Fixed priority scheduling involves two major subproblems to solve: (i) schedulability test; (ii) attribute assignment, e.g., priority and period assignment. Task τ_i may release its jobs periodically, but sometimes, a more flexible release policy is needed to meet complex timing constraints. We generalize periodic release of $\tau_{i,j}$ by a release algorithm, denoted by R_i. Release algorithm R_i returns the value $r_{i,j}$ when the system is on-line, and job $\tau_{i,j}$ is then released at this time. Or R_i may be implemented as a part of self-triggered task τ_i [14]. Thus, generally, attribute assignment involves the choice or development of algorithm R_i.

The general problem is to guarantee that $\lim_{j\to\infty} Y_{i,j} \geq Y_i^{Lo}$ for each hard real-time task τ_i; and to maximize $\lim_{j\to\infty} Y_{i,j}$ for each soft real-time task τ_i.

In [9] a special subclass of generalized hard timing constraints, called interval constraints and denoted by $\Xi_{i,j}$, is described. Thus xy-tasks model is specialized, i.e., the interval constraint $\Xi_{i,j}$ can be considered as an approximation of generalized hard timing constraint $G_{i,j}$. Fig. 5 can give the intuition about the definition of $\Xi_{i,j}$. Many timing constraints of control tasks can be represented as $\Xi_{i,j}$ [9]. The important property of $\Xi_{i,j}$ is that the set of all admissible $r_{i,j}$ is a continuous time interval for a given $\Xi_{i,j}$ and for $\forall j \geq 1$. For tasks with interval constraints, the schedulability test is proposed. This test is adapted for tasks with periodic release patterns. Also, several heuristic algorithms for period and priority assignment are developed for tasks with periodic and sporadic release patterns. These algorithms are evaluated using simulation of task scheduling with various timing constraints and other settings. Some results of these simulations are shown in [9]. The proposed model and algorithms could be implemented in a software tool for schedulability analysis. Particularly, some transformations and derivations of timing constraints, made in symbolic form, could be automated in such a tool.

IV. Open Problems

Problem 1: How to find the optimal (or at least, suboptimal) on-line release algorithm R_i or static release pattern for each task with $\Xi_{i,j}$ under fixed priority policy? Notice that priority assignment is also the problem to solve in this case.

Problem 2: How to efficiently schedule the mixture of soft and hard real-time tasks with $\Xi_{i,j}$? Notice that dynamic release algorithm R_i can be aware about soft real-time jobs awaiting in the queue, thus it can adjust the release time of each job accordingly. For example, some kind of Slack Stealing technique can also be applied.

Problem 3: Other approximations of generalized hard timing constraint $G_{i,j}$ should be investigated to efficiently resolve the trade-off between the expressiveness of the constraint and the complexity of scheduling, taking into account the practical value of such approximation.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[2] A. K. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," *Massachusetts Institute of Technology*, 1983.

[3] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE Transactions on Software Engineering*, vol. 23, no. 10, pp. 635–645, 1997.

[4] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, no. 1, pp. 5–22, 1999.

[5] N. T. Moyo, E. Nicollet, F. Lafaye, and C. Moy, "On schedulability analysis of non-cyclic generalized multiframe tasks," in *22nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2010, pp. 271–278.

[6] S. K. Baruah, "Dynamic-and static-priority scheduling of recurring real-time tasks," *Real-Time Systems*, vol. 24, no. 1, pp. 93–128, 2003.

[7] S. Baruah, "The non-cyclic recurring real-time task model," in *IEEE 31st Real-Time Systems Symposium (RTSS)*, 2010, pp. 173–182.

[8] M. Stigge, "Real-time workload models: Expressiveness vs. analysis efficiency," Ph.D. dissertation, Acta Universitatis Upsaliensis, 2014.

[9] M. Kavalerov, "Real-time xy-task model: Generalized control task model and its application to fixed priority scheduling," Department of Automation and Telemechanics, Perm National Research Polytechnic University, at_pstu_ru-rts-2017-01, URL: http://file.at.pstu.ru/materials/courses/rts/at.pstu.ru-rts-2017-01.pdf, Tech. Rep., 2017.

[10] K. W. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Systems*, vol. 6, no. 2, pp. 133–151, 1994.

[11] O. Redell and M. Sanfridson, "Exact best-case response time analysis of fixed priority scheduled tasks," in *14th Euromicro Conference on Real-Time Systems*, 2002, pp. 165–172.

[12] M. Kavalerov and N. Matushkin, "Obtaining the estimates of job execution parameters for fixed priority scheduling," Department of Automation and Telemechanics, Perm National Research Polytechnic University, at_pstu_ru-rts-2006-01, URL: http://file.at.pstu.ru/materials/courses/rts/at.pstu.ru-rts-2006-01.pdf, Tech. Rep., 2006.

[13] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *IEEE Real-Time Systems Symposium*, 2008, pp. 291–300.

[14] M. Velasco, J. Fuertes, and P. Marti, "The self triggered task model for real-time control systems," in *Work-in-Progress Session of the 24th IEEE Real-Time Systems Symposium (RTSS03)*, vol. 384, 2003.

Scheduling Interactive HPC Applications

Vladimir Nikolov, Stefan Bonfert
Inst. of Inform. Resource Management
Ulm University, Germany
Email: vladimir.nikolov@uni-ulm.de

Eugen Frasch, Franz J. Hauck
Inst. of Distributed Systems
Ulm University, Germany

I. Introduction and General Problem Overview

While modern real-time systems are commonly associated with embedded devices and limited computing power, they recently receive increased attention in the field of high performance computing (HPC) [1]. Many HPC applications perform some form of simulation, e.g. a weather forecast, fluid dynamics or heat dissipation. Typically, a user triggers these simulations with a set of input parameters and collects their output after they finished execution. However, interactive HPC simulations enable users to change parameters while the simulation is running and get immediate feedback. For example, the user might alter the shape of a car during an airflow simulation, or dynamically fine-tune its traction control system for certain simulated ground conditions. To enable interactivity, applications must be highly responsive and should meet tight timing constraints for a continuous user interaction loop, i.e. user input, intermediate processing and result visualization (e.g. with a certain frame rate).

HPC systems are typically realized as distributed clusters of throughput-optimized compute nodes with a high-speed interconnect and a job scheduler. Like in a typical batch system users submit individual jobs with a fixed reservation for *exclusive resources* (e.g. a set of compute nodes) for a certain time period. Modern schedulers like Slurm or Moab perform a simple cluster-wide admission control on job granularity, i.e. assign available resources to individual jobs. However, fixed reservations require a-priori knowledge about applications peculiarity, processing demands and completion time. HPC simulations in turn are compute intensive and highly parallel applications which naturally work in an iterative *fork-join* fashion (see Fig. 1). They periodically (or sporadically) spawn many parallel tasks which are distributed over the reserved resource set (fork). On completion they are synchronized at a common barrier (join) where partial results are gathered and analyzed. This procedure repeats with a dynamic data-preparation phase between a join and the next fork phase.

Technically, this execution model is currently supported by standards like MPI, for inter-process communication and data transmission, and OpenMP, for node-local task parallelism. However, from the perspective of interactive HPC simulations the model is insufficient. Assuming that user input will be incorporated and processed only during each data preparation phase, i.e. after a join and before the next iterative fork, a real-time execution model for the compute intensive parts is required. Thus, a common relative deadline for the parallel computations should be respected on every periodic/sporadic fork. Only then a certain processing rate and the required interactiveness of the simulation can be supported and guaranteed.

Obviously, the current fixed reservation model is also obsolete for interactive applications whose termination is now controlled by the user. Since traditionally resources are reserved exclusively on a per-job basis, compute nodes remain idle between every join and the next fork. A more sophisticated resource management strategy could allow for interleaved job execution on the same compute nodes, which would fill the waiting times and rise the node utilization factor and thus the throughput of the whole cluster. The respective strategy will be discussed in the following.

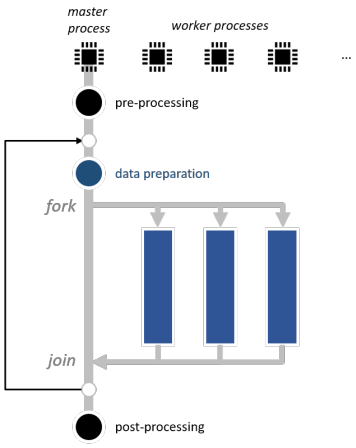


Fig. 1. Fork-Join Execution Model

II. Presupposition

The need for a general theory covering both high performance computing and real-time theory has been answered with the *real-time divisible load theory* (RT-DLT) [2]. It relies on a system model that closely resembles a typical HPC cluster setup, where one head node receives jobs from users, performs scheduling and distributes the workload on available compute nodes. For workload partitioning and distribution, RT-DLT relies on the classical *divisible load theory* (DLT) [3], that assumes a relation between the amount of data processed by a task and its execution time. Given a divisible job RT-DLT solves the scheduling problems of (a) how to divide the job among a set of processors in order to minimize its completion time, and (b) what is the minimum number of processors assigned to that job in order to meet its deadline. In the literature RT-DLT has been further elaborated for heterogeneous clusters and different processor ready times [4]. However, RT-DLT does not respect the typical iterative fork-join behaviour of HPC-applications. It considers only aperiodic workloads which are divided once, distributed to an exclusive set of resources and run to completion. In case of periodic job submissions conventional RT-DLT would lead to unnecessary re-scheduling, communication and task initialization phases on variable compute nodes.

III. Hierarchical Job Scheduling

We propose a hierarchical scheduling approach on top of RT-DLT for *periodic divisible jobs* $J_i = (A_i, T_i, \sigma_i, D_i)$, with a fixed inter-arrival time T_i (period), a maximal total data size σ_i per iteration and a relative deadline D_i. The basic idea is to sync the occurrence of periodic application forks with the activation sequence of its distributed tasks. However, assuming varying fork conditions as well as data preparation and transmission times, a sporadic model can be established as well. Figure 2 resembles the general scheduling architecture.

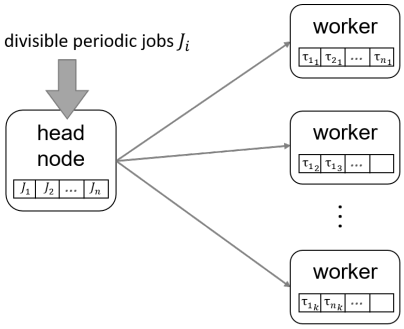


Fig. 2. Hierarchical Job Scheduling

Similar to RT-DLT a head node accepts and schedules jobs, but now several jobs may be hosted and executed at the same time on the same set of resources (*interleaved job execution*). When a job arrives and is accepted applying classical RT-DLT now relates to a particular job iteration. Following the rules in [3], [4] this results in the required number k_i of worker nodes for each job iteration in order to meet D_i and the minimal processing time C_i for the whole iteration (i.e., also for each node). Here for simplicity two assumptions were made:

- All nodes have homogeneous hardware; however, heterogeneous solutions for RT-DLT could be incorporated as well.
- The amount of processed data for each job iteration remains $\leq \sigma_i$, thus, C_i is a maximum estimation of iterations' processing costs; otherwise, we would have to reschedule the Job (i.e. recompute the RT-DLT) at each iteration.

Consequently, J_i then decomposes to a set of k_i periodic tasks of the form $\tau_{i,k} = (A_i, T_i, C_i, D_i)$ with equal periods T_i (equal to their job-period), costs C_i and relative deadlines D_i. These tasks can be now deployed to various worker nodes having local scheduling queues (cf. Fig. 2). Consequently, a node can host and execute several tasks of different or even the same job in parallel.

Let's assume the workers implement an EDF discipline, which allows an exact feasibility analysis for a node utilization factor up to 1 (i.e. 100%). As a result, the overall utilization U and the available slack time S can be computed for each particular node and its local task configuration. In case of $D_i \leq T_i$ a processor-demand–based feasibility test may even be used, e.g. approximation of tasks demand bound function [5]. Thereby, a synchronous start of the local task set can be assumed in oder to neglect actual activation phases of the tasks. However, meeting of $\tau_{i,k}$'s own deadlines, and thus the relative job deadline, is guaranteed on worker-node level as long as local feasibility can be proved. The node-specific utilization (or processor demand) and slack time are reported to the head node and used for further placement decisions within the cluster. These values have to be updated only if the local task configuration changes or their cost approximations are violated.

Let's assume, C_i are cost upper bounds for the periodic tasks and each node monitors its internal tasks' actual execution times; here a dynamic cost approximation model can be used like the one presented in [6]. If a task $\tau_{i,k}$ exceeds its current estimate C_i the actual schedule of the respective job J_i might be corrupted and meeting further iteration deadlines not guaranteed. In this case, the head node is required to compute a new schedule and redistribute the tasks if necessary.

IV. Discussion and Open Problems

It is obvious that with our hierarchical job scheduling model potential idle times of the workers and RT-DLT based rescheduling costs (i.e. job division, node initialization, communication) can be minimized. This model is particularly useful for interactive fork-join based parallel applications, which traditionally would occupy nodes idling during sequential application phases, data preparation and transmission (until the next fork). Even short blocking effects of tasks on the nodes can be masked locally with sensitive work of other jobs and tasks. Thus, the overall cluster utilization is increased while still being able to respect relative job deadlines, even for each fork-join phase.

Nevertheless, some problems still remain open for discussion. For node-local and cluster overload management we aim to extend our model with a dynamic quality control on task-level, like the one proposed in [6]. Here, quality decisions are opposed by potential task migrations and must apply in a distributed fashion. Apart from this some technical challenges exist like (a) how to sync data delivery to workers with the periods of their internal tasks, (b) how to capture worker local application state during task migrations, and (c) how to realize an optimal task placement, e.g. for minimal comm. costs on join phases.

References

[1] I. Alvarado, "Real-Time High-Performance Computing with NI LabVIEW," http://www.ni.com/white-paper/11972/en, 2012, [Online].

[2] X. Lin, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling for cluster computing," in *RTAS'07*, April 2007, pp. 303–314.

[3] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Computing*, vol. 6, no. 1, pp. 7–17, Jan. 2003.

[4] S. Chuprat and S. Baruah, "Scheduling divisible real-time loads on clusters with varying processor start times," in *Proceedings of the 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, ser. RTCSA '08. IEEE Computer Society, 2008, pp. 15–24.

[5] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *RTSS*, 1990.

[6] V. Nikolov, S. Wesner, E. Frasch, and F. Hauck, "A hierarchical scheduling model for dynamic soft-realtime systems," in *Proc. of the 29th Euromicro Conference on Real-Time Systems (ECRTS)*, 2017.

Increasing Fixed-Priority Schedulability using Non-Periodic Load Shapers

Mitra Nasri
Max Planck Institute for Software Systems (MPI-SWS),
mitra@mpi-sws.org

Geoffrey Nelissen
CISTER-INESC TEC, ISEP, Portugal.
grrpn@isep.ipp.pt

I. Motivations

Many real-time systems use *fixed-priority scheduling* (FPS) for three main reasons: (i) it is easy to understand, configure and analyse, (ii) it has low runtime overhead, and (iii) it is widely implemented in commercial operating systems. Additionally, FPS is enforced by some industrial safety related standards such as Autosar. However, FPS is not optimal from a schedulability viewpoint as it may prioritise the execution of non-urgent high-priority tasks over tasks with urgent deadlines.

As an attempt to increase FPS schedulability, variations of the FPS scheduling scheme such as Preemption Threshold Scheduling (PTS) [1] and Dual-Priroity (DP) scheduling [2] have been introduced. PTS raises the priority of a task (possibly several times) based on its remaining execution time, while DP promotes the task's priority after a given time following its release. It was shown that PTS is not optimal [3] and designing a strategy for correctly assigning priorities and promotion times such that all deadlines are met is still an open problem for DP [4].

In this work, we introduce *load-shaping* (LS); a technique that shapes the workload of tasks to limit their impact on the system's schedulability while improving the system's safety. LS is implemented using a reservation server for each task. Each server has a budget that replenishes itself following a given pattern.

Example. Fig. 1-(a) shows an example of a periodic task set which is not schedulable with FPS. However, the system may become schedulable if the workload released by τ_1 is properly shaped. An example of such shaping is shown in Fig. 1-(b), where τ_1's execution is divided in two segments. τ_1 is allowed to execute 4 time units in the first segment and 8 time units in the second.

Note that load-shaping is more generic then a simple period transformation [5]. Period transformation divides the tasks' periods into smaller segments of equal lengths, while load-shaping may generate segments of different lengths executing unequal workloads (see example above). We claim that period transformation may result in having more segments than actually required for the system to become schedulable, e.g., for Fig. 1's example, period transformation would need to reduce τ_1's period to 10 (i.e., the greatest-common-divisor of the tasks' periods), hence, dividing τ_1 in 3 segments instead of 2 as with load-shaping.

II. System Model and Notations

In this work, we consider a uniprocessor system executing a periodic task set τ. Each task τ_i in τ is characterised by a period T_i, a worst-case execution time (WCET) C_i, a relative deadline $D_i \leq T_i$, and a priority p_i (a lower value denotes a higher priority). Tasks are indexed such that $p_1 \leq p_2 \leq \ldots \leq p_n$. We assume that tasks are independent, do not share resources and do not self-suspend. The task set utilisation is denoted by $U = \sum_{i=1}^{n} C_i/T_i$.

A *shaper task* (or simply shaper) τ_i^S for an original task $\tau_i \in \tau$ is defined as a sequence of segments $\langle S_{i,1}, S_{i,2}, \ldots, S_{i,m_i} \rangle$, where each segment $S_{i,j} = (r_{i,j}, c_{i,j}, d_{i,j})$ is defined by a relative release time $r_{i,j}$ w.r.t. the release time of a job of τ_i, a relative deadline $d_{i,j}$ w.r.t. $r_{i,j}$, and a budget $c_{i,j}$ that limits the time during which τ_i can execute within segment $S_{i,j}$.

III. Open Problem

As explained earlier, parameters of the shaper tasks are needed in order to use load-shaping. In addition, a schedulability test is needed to ensure that all timing constraints will be met at run-time. The following problem summarises our main goal.

Problem 1. Given a task set τ and a schedulability test S, find a set of parameters for shapers such that τ becomes schedulable with S.

Note that a solution to Problem 1 always exists. Indeed, we know that by using period transformation and breaking the periods of the original tasks into segments that are equal to the greatest-common-divisor (GCD) of all periods and deadlines, then the shaped task set becomes harmonic. Hence, by using Rate Monotonic (RM) it is possible to schedule the shaped task set with no deadline miss if the total utilisation U is smaller than 1 [5]. However, this solution significantly increases the number of preemptions. Therefore, the actual open problem is the following.

Problem 2. Given an unschedulable task set τ with $U \leq 1$, how can we assign parameters to shapers such that the system becomes schedulable and the number of preemptions is minimised.

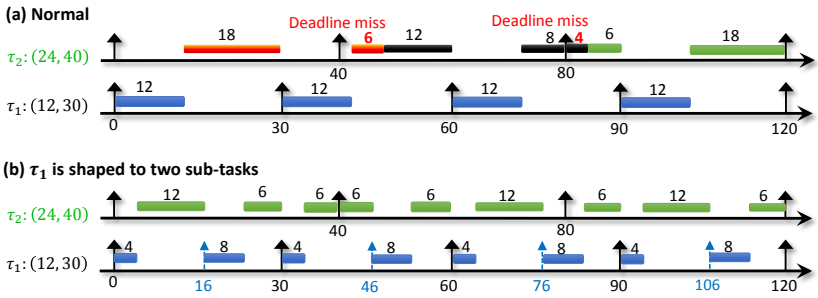


Fig. 1. An example of load shaping. In a normal execution (a), task τ_2 is not schedulable. Using a load shaper for τ_1 (b) composed of two segments $S_{1,1} = (0, 4, 16)$ and $S_{1,2} = (16, 8, 14)$, task τ_2 becomes schedulable (assuming implicit deadlines).

IV. Sketch of a Solution

Since each segment of a shaper task has a fixed release offset w.r.t. to the release time of its associated jobs, one can model segments as independent tasks that share the same priority and period than their task but have different release offsets and deadlines. Therefore, one can use existing schedulability tests for periodic tasks with release offsets (e.g., based on a feasability-interval) as the underlying response-time analysis.

An efficient solution to assign parameters to shapers and check the system's schedulability is to do both at the same time, and hence, find a set of parameters that guarantees schedulability by design. A constructive technique can be sketched as follows: **(i)** Start by assuming that each task τ_i has a default shaper $\tau_i^S = \langle S_{i,1} \rangle$, where $c_{i,1} = C_i$, $d_{i,1} = D_i$, and $r_{i,1} = 0$, **(ii)** Check the system's schedulability by analysing the response time of each job in a feasability-interval. Stop at the first unschedulable job J_i. **(iii)** Break the shapers of tasks with higher priorities than J_i into more segments such that the supply-bound-function (SBF) becomes larger than the request-bound-function (RBF) of level i in at least one time instant between the release and deadline of J_i. Go back to step (ii).

The challenge of step (iii) is to choose a time instant t (or maybe a set of such instants) at which the RBF is larger than the supply, and try to reduce RBF(t) by $\delta(t) = \max\{0, \text{RBF}(t) - \text{SBF}(t)\}$. For example, a potential candidate for t is a time instant at which $\delta(t)$ is minimized, since in that case, a smaller amount of higher-priority workload must be postponed. However, in the general case, one may attempt to postpone higher-priority workload for any t at which $\delta(t)$ changes its value. For each time instant t, there is a set of high priority tasks that are not finished yet. Among these tasks, those that have their deadline later than t are the candidates whose workload can be postponed.

V. Discussion

Beside safety-critical systems for which load-shaping allows higher predictability and dependability, applications of load-shaping can be found in multi-constrained systems, where the timing behaviour of a task might be constrained by other non-functional properties. Tasks might use different shapers depending on runtime requirements or system needs. For example, the shaper of a task τ_i can be different according to the following constraints: reduced response-time or reduced processor temperature. Under the first constraint, the shaper should assign more budget at the release of each job while in the latter case the shaper should force the task to be scheduled with a (potentially lower) rate leading to a better temperature profile for the system. The problem therefore becomes: how can one find shapers' parameters under additional constraints. Further, if a system has several modes of execution for the same set of tasks (e.g., power saving and high performance mode), one should understand how to switch between modes of one or more shapers without causing any deadline miss in the system.

Acknowledgment

This work is supported by a fellowship from Alexander von Humboldt Foundation. It was also partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the Portugal2020 Program, within the CISTER Research Unit (CEC/04234).

References

[1] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 1999, pp. 328–335.

[2] A. Burns and A. J. Wellings, "Dual Priority Assignment: A Practical Method for Increasing Processor Utilisation," in *Euromicro Workshop on Real-Time Systems (EWRTS)*, 1993, pp. 48–55.

[3] R. J. Bril, M. M. van den Heuvel, and J. J. Lukkien, "Improved feasibility of fixed-priority scheduling with deferred preemption using preemption thresholds for preemption points," in *International conference on Real-Time Networks and Systems (RTNS)*. ACM, 2013, pp. 255–264.

[4] A. Burns, "Dual Priority Scheduling: Is the Processor Utilisation bound 100%," in *International Real-Time Scheduling Open Problems Seminar (RTSOPS)*, 2010, pp. 3–5.

[5] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling." in *IEEE Real-Time Systems Symposium (RTSS)*, 1986, pp. 181–191.

Need for Reservation Servers with Constrained Deadlines

Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {daniel.casini, alessandro.biondi, giorgio.buttazzo}@santannapisa.it

I. Introduction

So far, all reservation servers have been defined and implemented using an implicit deadline, meaning that the server budget can be consumed anywhere within the server period (see [1], Chapters 5 and 6). However, recent results on semi-partitioned scheduling in multicore systems [2], [3] describe situations in which the server budget has to be consumed within a deadline smaller than the reservation period. Using semi-partitioned scheduling some reservations are statically allocated to processors, whereas others are splitted among multiple processors, thus involving a migration of the workload. When a reservation is partitioned between two cores, its budget Q_i is divided in two portions, $Q_i^{'}$ and $Q_i^{''}$, such that $Q_i = Q_i^{'} + Q_i^{''}$. An efficient method for splitting the budget is the C=D splitting scheme [4], according to which the budget of the second reservation is scheduled with relative deadline $D_i^{''} = Q_i^{''}$, whereas the first one is scheduled with deadline $D_i^{'} = T_i - D_i^{''}$. To apply such a splitting scheme when scheduling reservations, the corresponding servers must be designed to work under constrained deadlines.

This paper focuses on resource reservation in deadline-based systems and formulates the open problem of extending the Hard Constant Bandwidth Server [5] (H-CBS) to work with a deadline less than or equal to its period, in such a way to guarantee a desired reservation bandwidth within a bounded service delay. The H-CBS is based on *earliest-deadline first* (EDF) scheduling and is one of the most used algorithm for implementing resource reservation in deadline-based operating systems, also adopted in the Linux kernel to implement the SCHED_DEADLINE scheduling class [6]. It extends the Constant Bandwidth Server (CBS) [7] by introducing the concept of hard-reservation, according to which, once the budget is depleted, the server is suspended until the next replenishment time. In this manner, the *deadline-aging* problem [8] is avoided and the server execution becomes more regular. One of the main advantages of the H-CBS with respect to other reservation algorithms is to provide a **bounded** *maximum service delay* $\Delta_i = 2(T_i - Q_i)$, which is originated when the worst-case scenario illustrated in Figure 1 occurs. The *maximum service delay* Δ_i and the bandwidth α_i constitute an alternative interface for tuning the reservation parameters (Q_i, T_i). However, the algorithm was designed to work with an implicit-deadline D_i equal to its period T_i.

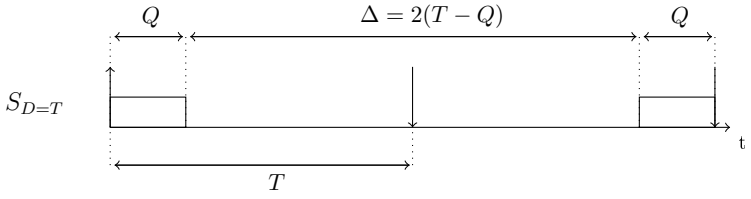


Fig. 1. Maximum service delay considering the H-CBS server, which assumes $D = T$. The worst-case scenario includes two consecutive instances of the server: in the first one, the server provides all its budget at the beginning of the instance, while in the second one the server provides all its budget as later as possible such that the server schedulability is not violated. If workload running upon the server (e.g., a task) arrives after the first instance is completed, then it experiences the maximum service delay of Δ time units.

The H-CBS provides a specific scheduling rule to guarantee the maximum service delay $\Delta_i = 2(T_i - Q_i)$, which deals with the scenario in which the server is idle and new pending workload to be executed arrives. When at time t a job arrives (i.e., the server starts having pending workload) and the H-CBS is idle, a replenishment time is computed as $t_r = d_i - \frac{q_i}{\alpha_i}$, where q_i is the server budget that is available at time t. Then, if $t < t_r$ the server is suspended until time t_r, where the budget is replenished and the absolute deadline is postponed to time $t_r + T_i$; otherwise, the budget is immediately replenished and the absolute deadline is postponed to $t + T_i$. The deadline assignment performed by such a rule exploits the notorious property of EDF scheduling that allows guaranteeing the server schedulability by only reasoning in terms of *bandwidths*.

Assuming constrained deadlines, this H-CBS rule *cannot be used anymore*: indeed, the replenishment time t_r (and the corresponding deadline postponement to $t_r + T_i$) strictly relies on the bandwidth of the server, which—as commonly known—is not enough to ensure the schedulability under EDF in the presence of constrained deadlines. Consequently, the maximum service delay of the worst-case scenario illustrated in Figure 1 (which would be $\Delta_i = T_i + D_i - 2Q_i$) cannot be guaranteed by the current H-CBS scheduling rules.

A naive solution for extending the H-CBS in the presence of constrained deadlines is to discard the budget when the server becomes idle and set the replenishment time as $t_r = kT_i$ with $k \in \mathbb{N}_{\geq 0}$. However, in this way the maximum service delay is

much worse than $T_i + D_i - 2Q_i$. In fact, consider the scenario shown in Figure 2. The server begins its execution in the first period with a pending workload of ϵ units of time, with ϵ arbitrary small. After executing ϵ units of budget, the server becomes idle and discards the budget. Then, immediately after, a new task arrives, but the server has just discarded its budget—so it must wait for the next period. Finally, in the worst-case the server might be scheduled at the end of the following period, thus resulting in a total delay $\Delta_i = T_i + (D_i - Q_i)$. Therefore, the following questions arise:

1) *How we can obtain a new algorithm H-CBS-CD (H-CBS with Constrained Deadline)?*
2) *How to modify the replenishment rules for obtaining a better maximum-service delay?*
3) *Is it possible to achieve a maximum service delay equal to $\Delta_i = D_i + T_i - 2Q_i$?*

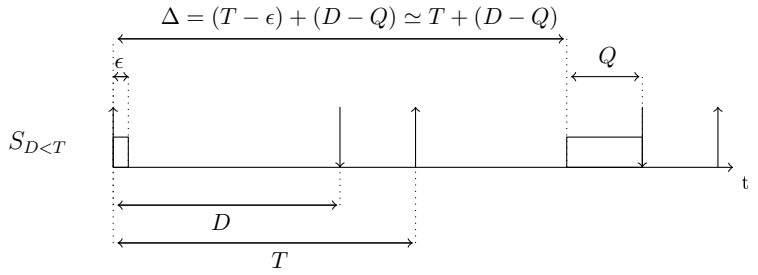


Fig. 2. Maximum service delay considering a naive solution for implementing H-CBS with $D < T$.

Other issues arise when considering resource sharing among tasks scheduled by different reservations. In this case, additional server rules are desirable to avoid budget exhaustions within critical sections, e.g., as done by the BROE [9] server (and in the corresponding version for multiprocessor platforms, M-BROE [10]). BROE extends the H-CBS and employs new scheduling rules that perform deadline postponements based on the server bandwidth: for the same reasons discussed above, such rules cannot be used in the presence of constrained deadlines. Then, another question arises: *4) How to guarantee a constrained-deadline bounded-delay partition in the presence of shared resources?*

Finally, the last issue to consider relies on how to perform admission control for new servers. Indeed, the reservation mechanism is often used in *open environments*, in which software components (implemented by servers) may join the system while the rest of the components continue to operate. Consequently, admission control has to be performed online. Considering implicit-deadline reservations, the admission can easily be done by checking $\sum_{r_i} \frac{Q_i}{T_i} \leq 1$ and then waiting for a safe time at which the new server can be admitted. Conversely, considering constrained deadlines, the schedulability test becomes more complex and will possibly be based on the *processor-demand criterion* (PDC) [11], which has exponential complexity in the worst-case. Then, it results to be unsuitable for being used online. A solution to this problem consists in approximating the PDC, for instance using the *fully polynomial-time approximation scheme* (FPTAS) proposed by Fisher et al. [12]. However, which admission control test is the most suitable for admitting a new reservation has still to be decided; also, it may be the case that a new test must be designed to simplify the on-line scheduling rules related to deadline postponements. Hence, another question arises: *5) Which admission control test should be used for admitting reservations?*

REFERENCES

[1] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Third Edition*. Springer, 2011.

[2] B. Brandenburg and M. Gul, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS 2016)*, Porto, Portugal, November 29 - December 2 2016.

[3] D. Casini, A. Biondi, and G. Buttazzo, "Semi-partitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing," in *Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, Dubrovnik, Croatia, 28-30 June 2017.

[4] A. Burns, R. Davis, P. Wang, , and F. Zhang, "Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme," *Real-Time Systems*, vol. 48, pp. 3–33, 2012.

[5] A. Biondi, A. Melani, and M. Bertogna, "Hard constant bandwidth server: Comprehensive formulation and critical scenarios," in *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, Pisa, Italy, June 18-20 2014.

[6] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016.

[7] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, Madrid, Spain, December 2-4 1998.

[8] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo, "Iris: A new reclaiming algorithm for server-based real-time systems," in *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)*, Toronto, Canada, May, 25-28 2004.

[9] M. Bertogna, N. Fisher, and S. Baruah, "Resource-sharing servers for open environments," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 202–219, July 2009.

[10] A. Biondi, G. C. Buttazzo, and M. Bertogna, "Supporting component-based development in partitioned multiprocessor real-time systems," in *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 8-10 2015.

[11] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-time systems*, vol. 2, no. 4, pp. 301–324, 1990.

[12] N. Fisher, T. P. Baker, and S. Baruah, "Algorithms for determining the demand-based load of a sporadic task system," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, Sydney, Australia, Aug, 16-18 2006.

A Problem of Time vs. Density Tradeoff in Multicore Fluid Scheduling

Kang-Wook Kim*, Jeongyoon Eo*, and Chang-Gun Lee*
*Department of Computer Science and Engineering, Seoul National University, Korea
Email: {kwkim, jyeo}@rubis.snu.ac.kr, cglee@snu.ac.kr

Abstract

Recent parallel programming frameworks such as OpenCL and OpenMP allow us to enjoy the parallelization freedom for real-time tasks. The parallelization freedom creates the time vs. density tradeoff problem in fluid scheduling, i.e., more parallelization reduces thread execution times but increases the density. By system-widely exercising this tradeoff, we propose optimal parameter tuning of real-time tasks aiming at maximizing the schedulability of multicore fluid scheduling. Our experimental study shows that the proposed approach well balances the time and the density, and results in up to 80% improvement of the schedulability.

I. Introduction

The advent of parallel programming, such as OpenCL and OpenMP, allows us to parallelize a real-time task in many different ways, which we call parallelization freedom. Due to this parallelization freedom, we can take a tradeoff between "time" and "density". That is, if we parallelize a task into a larger number of threads, the "time" for executing each thread becomes shorter. However, due to the parallelization overhead, the total computation amount of the task, i.e., the sum of thread execution times, becomes larger, which in turn makes the task's "density" larger, where the task density is defined as the task's total computation amount divided by its deadline.

Regarding this time vs. density tradeoff, there exist the "time bound" and the "density bound". First, we have to parallelize a task into a large enough number of threads such that each thread execution time becomes shorter than the given deadline. Otherwise, the deadline cannot be met in any way since each parallelized thread should be sequentially executed. Thus, the given deadline can be considered as an upper bound for the thread execution time. Second, a well known theory says that the ideal fluid scheduling [1] or its practical variations [2]–[4] can schedule all the threads of all the tasks before their deadlines if the system density, i.e., the sum of task densities, is smaller than or equal to the number of cores at any time. Thus, the number of cores can be considered as an upper bound below which the system density must be kept at all the times.

This paper tries to system-widely enjoy this tradeoff to maximize the schedulability of the multicore fluid scheduling [1]–[4]. For this, our approach is to minimize the peak system density under the time bound constraints by tuning four parameters of every task: (1) artificial period, (2) artificial deadline, (3) offset, and (4) parallelization.

II. Problem Description and Solution

We consider a system with M homogeneous CPU cores and N parallelizable periodic tasks. A parallelizable periodic task is defined as $\tau_i = (P_i, D_i, C_i(O_i))$ as shown in Fig. 1, where P_i is the period and D_i is the relative deadline. Also, O_i is the parallelization option indicating that the task τ_i is parallelized into O_i threads. O_i ranges from 1 to $O^{\max}$ meaning that we have options of parallelizing the task τ_i into single thread, i.e., $O_i = 1$, two threads, i.e., $O_i = 2$, and up to $O^{\max}$ threads, i.e., $O_i = O^{\max}$ as shown in the table of Fig. 1 (Fig. 1 shows an example case where the parallelization option $O_i = 2$ is chosen). Also, we use $e_i^k(O_i)$ $(1 \le k \le O_i)$ to denote the execution time of the k-th thread with the parallelization option of O_i and $C_i(O_i)$ to denote the total computation amount or simply "computation amount" of all O_i threads, i.e., $C_i(O_i) = \sum_{k=1}^{O_i} e_i^k(O_i)$. Out of the O_i thread execution times, the largest one is denoted by $e_i^{\max}(O_i) = \max_{1 \le k \le O_i} e_i^k(O_i)$. We assume that a larger parallelization option $O'(> O)$ makes the thread execution time smaller, i.e., $e_i^{\max}(O') < e_i^{\max}(O)$ but makes the computation amount larger, i.e., $C_i(O') > C_i(O)$, due to the parallelization overhead.

For scheduling these N parallelizable periodic tasks on M homogeneous CPU cores, we assume the ideal fluid scheduling algorithm, e.g., P-Fair [1] or its practical variations [2]–[4]. Under these assumptions, the given task set is schedulable if the following two conditions are met:

- **Time bound condition**: Each thread execution time of every task should be smaller than or equal to the given deadline, i.e, time bound. Otherwise, the given deadline cannot be met in any way since a thread should be executed sequentially.
- **Density bound condition**: In the fluid scheduling, a thread is defined to be *active* from its release time to its absolute deadline. Also, the density of an active thread is defined as its execution time divided by its relative deadline, i.e., the absolute deadline minus the release time. Thus, the *system density* at a time is defined as the sum of densities of threads

C.-G. Lee is the corresponding author. This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the "SW Starlab" (IITP-2015-0-00209) supervised by the IITP (Institute for Information & communications Technology Promotion).

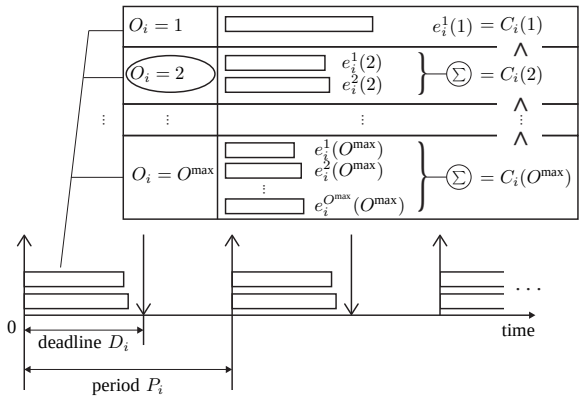


Fig. 1. Parallelizable periodic task model

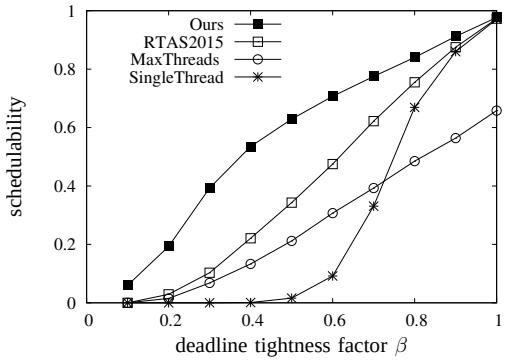


Fig. 2. Schedulability according to the deadline tightness factor β

which are active at that time. With these definitions, the system density at any time should be lower than or equal to the number of cores, i.e., density bound. Otherwise, some threads cannot meet their deadlines by the fluid scheduling.

The problem we tackle in this paper is how to system-widely exercise this tradeoff to maximize the schedulability. For this, we have four controllable parameters, i.e., the task offset Φ_i, the parallelization option O_i, the artificial deadline $d_i(\leq D_i)$, and the artificial period $p_i(\leq P_i)$ for every tasks τ_i. For a task τ_i, its density function along time t is formulated as $\delta_i((\Phi_i, O_i, d_i, p_i), t)$. With this density function of each task τ_i, our system-wide time vs. density tradeoff problem can be formulated as the problem of minimizing the peak of the system density, i.e., the sum of all N task density functions, within the hyper period HP while satisfying the time bound condition for each thread: minimize $\max_{0 \leq t < HP} \sum_{i=1}^{N} \delta_i((\Phi_i, O_i, d_i, p_i), t)$. Our proposed solution approach to optimally exercising the four control knobs is firstly optimally determine d_is, O_is, and Φ_is for a task group with the same period. Using this as a building block method, an iterative heuristic algorithm finds a near-optimal solution, (Φ_i, O_i, d_i, p_i) for all τ_is, that minimizes the peak system density.

III. Experimental Result

We conduct simulation studies to investigate the schedulability improvement by the proposed algorithm. For this, the task set Γ is formed with N synthetic tasks. The deadline D_i of τ_i is determined using a so called "deadline factor" β as $D_i = \beta \cdot P_i$. Such generated N tasks in Γ are scheduled using M homogeneous CPU cores, where $M = 8$. In order to show the schedulability improvement, we compare our approach marked as "Ours" with other three approaches. With the approach marked as "SingleThread", every task is executed with a single thread without parallelization. With the approach marked as "MaxThreads", every task is parallelized into the maximum number of threads. The approach marked as "RTAS2015" is the approach in [5] where each task uses the optimal parallelization option for the given D_i individually, but not system-widely.

Fig. 2 shows the schedulability of "Ours" and other three approaches as ranging the deadline factor $\beta = D_i/P_i$ from 0.1 to 1. When β is small, that is, when the time bound is tight, "SingleThread" cannot schedule most of task sets. This is because the thread execution time overflows the time bound D_i most of times even when the peak system density is smaller than the density bound $M = 8$. On the other hand, "MaxThreads" can schedule more task sets since the thread execution times using the maximum parallelism can be smaller than the time bound D_i. However, as increasing β, the situation reverses. "SingleThread" that gives favor to minimizing the density using a single thread has much higher schedulability than "MaxThreads" that uses large densities for using the maximum parallelism. "RTAS2015" trades off the time and the density for each task individually and hence gives higher schedulability than "SingleThread" and "MaxThreads" in the entire range of β. "Ours" gives even higher schedulability since it takes the system-wide tradeoff of time and density considering all the tasks together. As a result, when $\beta = 0.5$, "Ours" shows 80%, 191%, and 3750% higher schedulability than "RTAS2015", "MaxThreads", and "SingleThread", respectively.

References

[1] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate Progress: a Notion of Fairness in Resource Allocation," in *25th annual ACM symposium on Theory of computing*, 1993.

[2] A. Srinivasan and J. H. Anderson, "Fair Scheduling of Dynamic Task Systems on Multiprocessors," *Journal of Systems and Software*, vol. 77, no. 1, pp. 67–80, 2005.

[3] S. Funk, "LRE-TL: An Optimal Multiprocessor Algorithm for Sporadic Task Sets with Unconstrained Deadlines," *Real-Time Systems*, vol. 46, no. 3, pp. 332–359, 2010.

[4] G. Nelissen, V. Berten, V. Nélis, J. Goossens, and D. Milojevic, "U-EDF: An Unfair but Optimal Multiprocessor Scheduling Algorithm for Sporadic Tasks," in *24th Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[5] J. Kwon, K.-W. Kim, S. Paik, J. Lee, and C.-G. Lee, "Multicore scheduling of parallel real-time tasks with multiple parallelization options," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2015.

Orthogonal to Multiprocessor Resource Sharing Protocols: How to Share?

Jian-Jia Chen
Department of Informatics, TU Dortmund University, Germany

I. Introduction

In a multi-tasking system, mutual exclusion for the accesses to shared resources, e.g., data structures, files, etc., has to be guaranteed to ensure the correctness of the operations, especially under the von-Neumann programming model. Such accesses to shared resources are typically done within the so-called *critical sections*, which can be protected by using semaphores. Due to the mutual exclusion, it is unavoidable that a higher-priority task may be blocked by a lower-priority task, if aborting and restarting a critical section are not allowed. In uniprocessor systems, locking protocols based on *priority inheritance*, such as the priority ceiling protocol (PCP), the priority inheritance protocol (PIP), and the stack resource policy (SRP), have been proposed to handle the priority inversion problem reasonably well.

For real-time tasks on multiprocessor platforms, the design of synchronization protocols started with the distributed priority ceiling protocol (DPCP) [13], followed by the multiprocessor priority ceiling protocol (MPCP) [12]. Under the DPCP, the tasks are scheduled based on partitioned fixed-priority (P-FP) scheduling, except when accessing resources that are bound to a different processor. By constrast, the MPCP adopts the PCP for local resources, but when requesting global resources tasks execute the corresponding critical sections on their assigned processors. To reduce the blocking time of global critical sections, the priorities of global critical sections are boosted under the DPCP and MPCP. In the original design of the DPCP and MPCP, a task *suspends* itself while being blocked on the global shared resource. By contrast, a task can also perform busy waiting (under spin-based protocols such as the MSRP [7]) while being blocked on shared resources. *Busy waiting* has been shown to be efficient when critical sections are short, but the resulting loss of processor service must be accounted for. More recently, Burns and Wellings [4] proposed a variant of the MSRP, the Multiprocessor resource sharing Protocol (MrsP) [4], to exploit the spinning cycles of one task to help other tasks make progress.

Over the years, many suspension-based locking protocols and spin-based locking protocols have been designed and analyzed, but only very few results are available for the counter-part of the problem: *How to partition and how to share?* For all the protocols mentioned above, it is known that the number of priority-inversion blockings can be lower bounded by the number of processors of the multiprocessor system in the worst case, for specific task partitions, as shown in [3]. That is, the elegance of partitioned scheduling to run a task all the time on one processor suffers from the synchronization between the real-time tasks if the tasks are not partitioned well.

Essentially, the performance of resource sharing protocols highly depends on how the tasks are partitioned. With regard to task partitioning, the following results have been reported: Lakshmanan et al. [10] presented a synchronization-aware partitioned heuristic tailored to the MPCP, which organizes tasks sharing common resources into groups and attempts to assign each group of tasks to the same processor. Following the same principle, Nemati et al. [11] presented a blocking-aware partitioning method that uses advanced cost heuristic to split task group when an entire group fails to be assigned on one processor. Hsiu et al. [8] proposed a dedicated-core framework to separate the execution of critical sections and normal sections, and employed an priority-based RPC-like mechanism for resource sharing, such that each higher-priority request can be blocked by at most one lower-priority request. Unfortunately, there were some issues related to the adopted schedulability tests in the above results. Please refer to [5], [15] for details. More recently, Wieder and Brandenburg [14] used integer linear programming and developed a Greedy Slacker (GS) algorithm based on Audsley's priority assignment to solve the partitioning problem in the presence of spin locks based on MRSP. However, these results are heuristic algorithms, and there has been no algorithmic analysis provided in the literature to necessitate these approaches from the perspective of theoretical optimality guarantees, e.g., based on resource augmentation (or speedup) factors. Moreover, there is no clear evidence showing whether the above locking protocols designed to follow the traditional partitioned scheduling paradigm.

Andersson and Easwaran [1] presented an virtualization-based G-EDF (global earliest-deadline-first) scheduling with shared resources, which guarantees a $12(1 + 3r/(4m))$ speedup factor on a platform comprising m identical processors, where each task uses at most one of the r shared resources and each job may request the resource at most once. Andersson and Raravi [2] extended the virtualization-based scheduling and achieved a speedup factor of $4 \cdot (1 + \lceil \frac{r}{m} \rceil) \geq 8$. Huang et al. [9] proposed resource-oriented partitioned PCP, called *ROP-PCP*, that has a speedup factor of $11 - \frac{6}{m+1}$. *ROP-PCP* uses two dedicated subsets of the m processors to execute the critical and the non-critical sections individually.

II. Open Problems

Although many multiprocessor resource sharing protocols based on partitioned scheduling, including the DPCP, MPCP, MSRP, MrsP, etc., have been designed in the literature, their performance is highly dependent on how the tasks are partitioned. Without a suitable task partitioning algorithm for a specific protocol, the performance of the protocol may be over-estimated or under-estimated. Among all the existing locking protocols, ROP-PCP [9], based on the DPCP, is the only one that has a bounded augmentation factor by using an associated pseudo-polynomial-time task partitioning algorithm. The approaches in [1], [2] used nonpreemptive EDF scheduling within a shared resource based on virtualization. In my view, the following open problems are of importance and of interest for the future design and analysis of multiprocessor resource sharing protocols:

- *To fairly compare these protocols, partitioning algorithms elegantly designed for the MPCP, MSRP, and MrsP are needed.* The Synchronization-Aware Partitioning Algorithm (SPA) proposed by Lakshmanan et al. [10] is an algorithm which partitions tasks under the MPCP. Informally speaking, SPA tries to allocate the tasks that request the same resource on one processor if possible. However, if the utilization of these tasks is more than 100%, this approach needs to split these tasks and does not work very well. It is rather easy to apply SPA under MrsP. However, the experimental results by Huang et al. [9] showed that SPA does not derive good task partitions for the MrsP and MPCP protocols. The Block-Aware Partitioning Algorithm (BPA) proposed by Nemati et al. [11] used two rounds for partitioning by sorting the tasks according to a weighted function defined based on the utilization and the blocking time divided by the period. The Greedy Slacker (GS) algorithm by Wieder and Brandenburg [14] "is oblivious to the choice of locking protocol and uses an intriguingly simple greedy approach." Therefore, the GS is also applicable for the MPCP and MrsP. It is also possible to integrate the schedulability tests and the task partitioning problem as a mixed-integer linear programming (MILP), e.g., the MILP by Wieder and Brandenburg [14] for MSRP. However, the scalability is a concern.
- Most of the schedulability analyses of these protocols are applicable for constrained-deadline task systems. However, most of the partitioning algorithms only focused on implicit-deadline task systems. *Partitioning algorithms elegantly designed for constrained-deadline task systems are needed.* In the ROP-PCP approach by Huang et al. [9], the critical sections guarded by one semaphore are assigned to processors first. This step requires to sort the semaphores according to the critical-section utilizations. Therefore, extending to constrained-deadline task systems would need a more elegant approach. In the approaches in [1], [2], the deadline assignments can be applicable, but the speed/bandwidth of a virtualized synchronization processor for a semphaore has to be decided carefully. However, the speedup bounds in [1], [2] are not constants. Moreover, Chen et al. [6] demonstrate empirically that the stringent relative deadlines assignments in [1], [2] have significant drawbacks in terms of performance against the ROP-PCP [9].
- In [9], a task can have multiple critical sections, whilst a task can have only one critical section separated by two non-critical sections in [1], [2]. *Do the patterns of the critical sections significantly affect the difficulty for designing good partitioning algorithms?* In the presence of multiple resource accesses and multiple requests on one resource, ROP-PCP is the only result with bounded speedup factors.

References

[1] B. Andersson and A. Easwaran. Provably good multiprocessor scheduling with resource sharing. *Real-Time Systems*, 46(2):153–159, 2010.

[2] B. Andersson and G. Raravi. Real-time scheduling with resource sharing on heterogeneous multiprocessors. *Real-Time Systems*, 50(2):270–314, 2014.

[3] B. Brandenburg and J. Anderson. Optimality results for multiprocessor real-time locking. In *Proceedings of the 31st Real-Time Systems Symposium*, pages 49–60, 2010.

[4] A. Burns and A. J. Wellings. A schedulability compatible multiprocessor resource sharing protocol - MrsP. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 282–291, 2013.

[5] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Faculty of Informatik, TU Dortmund, 2017. 2nd Version.

[6] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and R. I. Davis. On the pitfalls of resource augmentation factors and utilization bounds in real-time scheduling. In *ECRTS*, 2017.

[7] P. Gai, G. Lipari, and M. D. Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Real-Time Systems Symposium (RTSS)*, pages 73–83, 2001.

[8] P. Hsiu, D. Lee, and T. Kuo. Task synchronization and allocation for many-core real-time systems. In *International Conference on Embedded Software, (EMSOFT)*, pages 79–88, 2011.

[9] W.-H. Huang, M. Yang, and J.-J. Chen. Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share? In *2016 IEEE Real-Time Systems Symposium, RTSS*, pages 111–122, 2016.

[10] K. Lakshmanan, D. De Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *RTSS*, pages 469–478, 2009.

[11] F. Nemati, T. Nolte, and M. Behnam. Partitioning real-time systems on multiprocessors with shared resources. In *Principles of Distributed Systems - International Conference, OPODIS*, pages 253–269, 2010.

[12] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings.,10th International Conference on Distributed Computing Systems*, pages 116 – 123, 1990.

[13] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, pages 259–269, 1988.

[14] A. Wieder and B. B. Brandenburg. Efficient partitioning of sporadic real-time tasks with shared resources and spin locks. In *International Symposium on Industrial Embedded Systems, (SIES)*, pages 49–58, 2013.

[15] M. Yang, J.-J. Chen, and W.-H. Huang. A misconception in blocking time analyses under multiprocessor synchronization protocols. *Real-Time Systems*, 53(2):187–195, 2017.

List of Authors