# ECRTS 2018

# Proceedings of the Work-in-Progress Session

Barcelona, Spain, July 4th
Edited by Martina Maggio

# Message from the Chair

Welcome to the Work in Progress (WiP) Session of the 30$^{th}$ Euromicro Conference on Real-Time Systems (ECRTS), held in Barcelona on July 4$^{th}$, 2018. This session provides an opportunity for researchers to present their ongoing work and get feedback that will help them in identifying promising future research, and in gaterhing new insights on their research topics.

The program, this year, contains 10 papers, on multiple aspects of Real-Time Systems, spanning from mixed criticality scheduling to cache-aware scheduling, and from operating systems to demand-bound functions. Each paper describes, in three pages, the current research status and future potential directions. I wish to all the authors to find inspiration, collaborators, and a forum to openly discuss new ideas.
The Session proceedings are available online, at the address

`https://erts18.ecrts.org`

selecting the menu item "Work in Progress Session". This collection of articles is the result of the work of many authors, that should be acknowledged for their contribution. Additionally, I would like to thank the members of the Technical Program Committee, that have done an impressive job at providing meaningful and constructive comments in a very limited time.
On behalf of the Program Committee, I wish you a pleasant and fruitful session. I hope you find the program inspiring and that you will take the opportunity to look at the poster and talk with the authors.

Martina Maggio
Lund University
ECRTS 2018 Work-in-Progress Chair

# Organizers

- **Martina Maggio**, Lund University, Sweden

## Program Committee

- **Arne Hamann**, Robert Bosch Gmbh, Germany
- **Patrick Meumeu Yomsi**, CISTER/INESC TEC and ISEP, Portugal
- **Rodolfo Pellizoni**, University of Waterloo, Canada
- **Marcus Völp**, University of Luxembourg, Luxembourg

# ECRTS 2018 Work-in-progress Papers

# Handling Intra-Task Parallelism for Real-Time DAG Tasks Scheduled on Multiple Cores

Johnathon Soulis*, Jaewoo Lee†, and Chang-Gun Lee*
*Department of Computer Science and Engineering, Seoul National University, Seoul, Korea
†Department of Industrial Security, Chung-Ang University, Seoul, Korea
Email: {taylor, cglee}@rubis.snu.ac.kr, jaewoolee@cau.ac.kr

***Abstract*—In this paper, we discuss a method for scheduling a set of parallel tasks modeled by a generic DAG task model. We present a method that handles the intra-task parallelism between computational units as well as the parallelization of individual computational units in order to schedule the tasks on multiple CPU cores. We introduce a density packing problem that describes our approach for handling intra-task parallelism in order to minimize the overall task minimum peak density.**

## I. Introduction

Compared to traditional real-time applications, recent real-time applications such as sensor processing, vision processing and deep learning neural networks [2] require more computation bounded requirements and have complicated task model structures such as multi-segment [3] and DAG [4] task models. Following this trend, even simple and cheap embedded boards have recently adopted multi-core CPU architectures [5] to support the parallel execution of tasks. For real-time tasks requiring heavy computation and modeled by complicated structures such as the one shown in Figure 1, methods to parallelize and schedule these tasks on multi-core processors are needed.

An optimal algorithm for parallelizing and scheduling a set of parallel tasks with multiple parallelization options on multiple CPU cores is presented in [1]. This paper addresses the scheduling problem as a density minimization problem, seeking to minimize the peak task set density of a task set described by a multi-segment task model. While this paper addresses scheduling multiple parallel tasks and determines the parallelization options for task segments during the scheduling process, it does not consider tasks in which intra-task segments are run in parallel. In other words, it only considers a multi-segment task model in which each task is represented as a sequential series of executable units. However, there is need to consider tasks in which computational units within a task itself can be scheduled to be run in parallel. This introduces a new layer of complexity to the scheduling problem addressed by [1]. For tasks that can be represented by a directed acyclic graph (DAG), we must determine how we will group or arrange parallel segments and how we will determine the density of these grouped segments.

In this paper, we address this scheduling problem of assigning parallelization options as a part of the scheduling process for tasks that can be represented by a DAG task model.

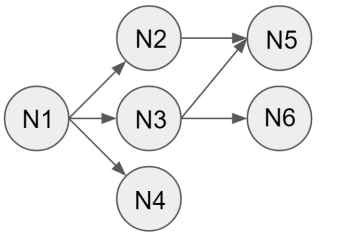



Fig. 1: Automotive task described by DAG task model

## II. Problem & Approach

As mentioned, an optimal algorithm for parallelizing and scheduling a set of parallel tasks with multiple parallelization options assuming a multi-segment task model underlies the tasks of interest has been introduced. Each segment $S_k$ of the multi-segment task has $p$ parallelization options such that a possible option for the segment could be represented as $Opt_k = p$. For example, consider that segment $S_1$ has parallelization option $Opt_1 = 2$ for task $\tau_i$. This means that segment $S_1$ of $\tau_i$ is parallelized into 2 threads. As $p$ increases and the number of threads increases, the time it takes to execute a given segment decreases and thus the segment can be completed by a shorter deadline. However, because the execution of the segment has been divided across multiple threads, the computational resources it takes to complete a segment that has been heavily parallelized also increases. In other words, The *density* of a segment with $Opt_k = p_b$ is greater than a segment where $Opt_k = p_a$ if $p_b > p_a$. This is because, as the parallelization of a segment increases, the parallelization overhead increases as well.

In this paper, we consider a a set of *n* independent tasks running on a system with *m* CPU cores. Each task $\tau_i$ is a sporadic task released repeatedly with a minimum inter-release time $T_i$ and should be completed by its relative deadline $D_i$. We assume the deadline is taken to be equal to the minimum inter-release time, $D_i = T_i$. Each task $\tau_i$ will execute a program that can be parallelized and the task will be represented by a DAG task model. Each node of the DAG can be thought of as

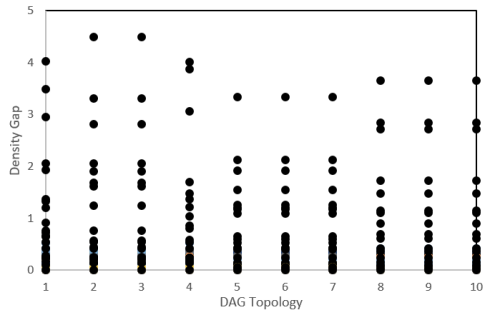



Fig. 2: Gap in density for good and bad handling of intra-task parallelism

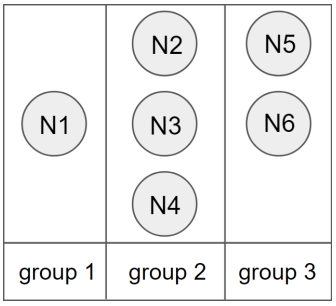



Fig. 3: Base transformation by observing precedent constraints

a computational unit that carries out a specific function or role for task $\tau_i$. Also, each node of the DAG can be parallelized into multiple threads. These threads can be executed independently. We assume that the thread executions on CPU cores can be preempted and migrated at any time with negligible cost.

The DAG task model presents a new challenge compared to the multi-segment task model in that we must consider the intra-task parallelism that exists between nodes of a DAG that can be executed in parallel. Our first approach to address the intra-task parallelism of nodes in a DAG was to explore the different ways in which nodes of the DAG could be grouped and serialized while satisfying precedent constraints between nodes. A naive solution that can be quickly imagined is to serialize the nodes of the DAG entirely so that the DAG task model reduces into a multi-segment task model. The scheduling methodology of [1] for multi-segment task model can then be applied directly. However, given a specific DAG topology, like the one shown in Figure 1, which depicts an example automotive task, we can generate not only this naive solution but also an exhaustive list of all possible ways to group and serialize nodes while preserving precedent constraints between nodes. We can then apply the scheduling method used for the multi-segment task model to each serialization to determine the minimized peak task density possible for the given serialization. While an exhaustive search for all possible ways to group nodes is not an ideal long term solution because the number of serializations possible grows exponentially as the number of nodes increases, it reveals insight into how handling the intra-task parallelism of the DAG poorly can result in a high task density that could have otherwise been avoided if the intra-task parallelism was handled differently.

We conducted an experiment to quantify how the handling of the intra-task parallelism has an effect on the peak task density of the task. Given a DAG topology we generated an exhaustive list of all the serializations. We then calculated the min peak task density on all the different serializations 100 times, varying the starting task parameters each time. Finally, we found which serializations the task experienced the lowest density compared to other serializations and the serialization it experienced the largest density. The difference of these two densities was then taken to represent the density gap experienced between good and poor handling of the intra-task parallelism. We performed this experiment in entirety for 10 different DAG topologies.

As we can see from Figure 2, the way in which we handle intra-task parallelism for a task can have huge impacts on the min peak density and subsequent schedulability of the task. Therefore it is important to have a good strategy in place for handling intra-task parallelism for the DAG task model. Motivated by these findings, we present our current work in progress and envisioned solution regarding scheduling tasks described by a DAG task model.

## III. Envisioned Solution

Our envisioned solution for this research is as follows:

1) Generate the *base transformation* for the DAG topology
2) Compute the *approximate density graph* for each node
3) Pack *density blocks* in way to minimize peak task density

We first generate the *base transformation* for the DAG topology. Given a task represented by a DAG like the one shown in Figure 1, we first generate our base transformation as shown in Figure 3. The base transformation puts an order on the nodes such that nodes in group $i+1$ have all of their predecessors in groups $i$, $i-1$, and so on.

Next we must compute the *approximate density graph* for each node, $n$. The *approximate density graph* will show us all of the density values that a certain node can take on for every parallelization option and possible virtual deadline for a given node. The density values calculated will be a an over estimation of the actual density values possible for a given deadline and parallelization option. This overestimation is due to the linear approximation we use to calculate the density values for each node. We are considering an approximate version of the density graph to make downstream calculations simpler. Especially calculations that involve the parallel or sequential alignment of nodes when producing the final schedule. This is essential as the number of calculations made with this structure in the case of the DAG task model is much more than the multi-segment task model considered previously.

The *approximate density graph* can be found fairly easily. First, we consider the cases in which the parallelization option for a node equals 1 and the max value for the number of threads, $Opt_n = 1$ and $Opt_n = MAX$ respectfully. For

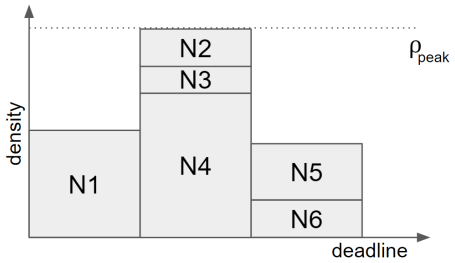



Fig. 4: An example of density blocks for a given DAG

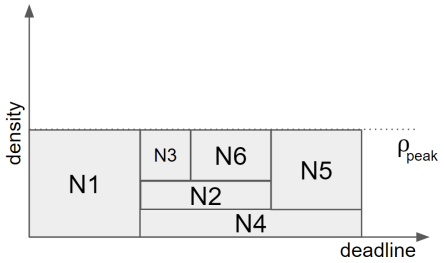



Fig. 6: An improved packing solution

each of these parallelization options we must calculate the execution time for each node, $E_n^{opt}$, as well as the sum of the individual thread execution times, $C_n^{sum,opt}$, so that we can calculate the density of the task. In [1], to generate the density graph, all virtual deadlines, $V_n$, such that $V_n \geq E_n^{opt=MAX}$ but $V_n < D_i$, overall task deadline, were considered. For each $V_n$, the density was calculated using $E_n^{opt}$. The resulting density graph is similar to that shown in Figure 5 (a). For our simplified approach, we will calculate the density $\rho^{high}$ when $Opt_n = MAX$ and $V_n = E_n^{opt=max}$, because this is when the execution time of the node is shortest and the density highest. When $Opt_n = 1$ and $V_n = E_n^{opt=1}$, the density will be 1, $\rho_n = 1$. This is because $C_n^{sum,opt=1} = E_n^{opt=1}$. With these points, $(V_n = E_n^{opt=MAX}, \rho_n^{high})$ and $(V_n = E_n^{opt=1}, \rho_n = 1)$, we can approximate the minimum density values that the node can achieve for parallelization options $Opt_n = MAX$ to $Opt_n = 1$ when $V_n \leq E_n^{opt=1}$. Lastly, we must approximate the varying density values the node can take when $Opt_n = 1$ and $E_n^{opt=1} < V_n \leq D_i$. This can be done by calculating the minimum density of the task $\rho_n^{low}$, which occurs when $V_n = D_i$. With these three points, $(V_n = E_n^{opt=MAX}, \rho_n^{high})$, $(V_n = E_n^{opt=1}, \rho_n = 1)$ and $(V_n = D_i, \rho_n^{low})$, we can approximate all of the minimum density values that a node can take on for any parallelization option and virtual deadline value. This graph is shown in Figure 5 (b).

Once we have generated an *approximate density graph* for each node, we can easily calculate the density for a node at a given virtual deadline and *vice versa*. These graphs allow us to imagine that each node is a block of density that can be arranged until we find a packing arrangement that results in the lowest density. For example, starting from our *base transformation* in Figure 1, we calculate the density of each node by assigning a virtual deadline at the end of each group. In that way, nodes in the same group share the same virtual deadline. We then are able to come up with something as shown in Figure 4. From this starting point, we can see how we might be able to move blocks around as long as we adhere to precedence constraints outlined by Figure 1. For example, we can see that node 4 does not have any successors and as a result can be run over a longer virtual deadline in order to reduce the impact it has to increasing the overall task density. We can see how after considering precedent constraints, different virtual deadlines and parallelization options for different nodes, we are left with a packing problem. Figure 6 shows an example solution of how we can improve on the baseline introduced in Figure 4.

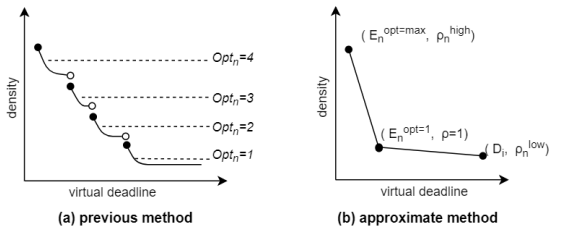



Fig. 5: Example density graph for one node

## IV. Conclusion

In this paper we introduced a method for scheduling real-time tasks described by a DAG task model with parallelization option freedom for each node. We emphasized the need of handling intra-task parallelism by introducing a density block packing problem as a model for determining how to schedule nodes. By observing the precedent constraints of nodes and the *approximate density graphs* of nodes, we can pack and re-arrange nodes in order to create a schedule with minimal peak task density. We are currently working on the implementation of this design and researching efficient ways to solve the density packing problem.

## References

[1] Jihye Kwon, Kang-Wook Kim, Sangyoung Paik, Jihwa Lee, and Chang-Gun Lee. *Multicore scheduling of parallel real-time tasks with multiple parallelization options*. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015.

[2] Young Jong Mo, Joongheon Kim, Jong-Kook Kim, Aziz Mohaisen, and Woojoo Lee. *Performance of deep learning computation with TensorFlow software library in GPU-capable multi-core computing platforms*. Ninth International Conference on Ubiquitous and Future Networks(ICUFN), 2017.

[3] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. *Multi-core real-time scheduling for generalized parallel task models*. Real-Time Systems, 49(4):404-435, 2013.

[4] Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. *Parallel Real-Time Scheduling of DAGs*. Real-Time Systems, 25(12):3242-3252, 2014.

[5] Intel, *Multicore Scalability for Embedded Systems*. http://www.embeddedintel.com/special_features.php?article=2241.

# Towards Optimal Offline Scheduling for Multi-Core Systems with Partitioned Caches

Darshit Shah*, Jan Reineke**
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
Email: *s8dashah@stud.uni-saarland.de, **reineke@cs.uni-saarland.de

***Abstract*—Cache partitioning is often used in multi-core systems to improve predictability. Prior work has mostly focused on per-core cache partitioning. We conjecture that there is significant remaining potential if the cache is partitioned at the granularity of individual jobs. In this paper, we discuss our ideas for a new algorithm—an extension of the A* algorithm—to find optimal schedules for non-recurrent sets of jobs.**

## I. Application Domain and Challenge

To address size, weight, and power constraints (SWaP) there is a trend to transition from federated architectures, where each application is deployed on a private single-core processor, to integrated architectures, where multiple applications share a single multi-core platform. In safety-critical real-time systems this trend poses a major challenge for the verification process due to the possibility of interference on shared resources, such as buses, networks-on-chip, shared caches, and memory controllers. Safely and tightly bounding this interference is extremely difficult as the shared resources and their hardware management mechanisms have usually not been designed with predictability in mind.

A rather general approach to address this challenge is to partition the shared resources in space and time. By partitioning the resources, interference can be eliminated, making the use of multi-core platforms feasible. Such an approach is also advocated in the recent position paper on certification issues CAST-32A [1], where it is referred to as *robust partitioning*. In this work, we focus on the challenge of efficiently partitioning shared caches.

## II. Motivation

Caches are a major source of unpredictability and pessimism in the analysis of multi-core hard real-time systems. This is because it is usually difficult to know the exact memory behavior of the tasks running simultaneously with the task under analysis. There are multiple ways of dealing with the issues of unpredictability due to a shared cache. Approaches that deal with a shared cache in a multi-core system, do so in one of the two following ways:

1) Cache Partitioning: These approaches [2]–[4] partition the shared cache using either hardware or software techniques. In this way they achieve temporal isolation between different jobs, thereby easing timing verification.
2) Analyzing Cache Interference: Other approaches [5]–[8] compute bounds on the interference between different jobs on a shared cache and use these bounds within response-time analysis.

Precisely analyzing cache interference is very challenging, as very different cache states and cache behaviors may arise, depending on the precise relative timing of memory accesses coming from different cores. The problem is slightly easier if jobs are scheduled non-preemptively. Still, even analyses assuming non-preemptive scheduling [8] make use of coarse abstractions of the cache behavior, and thus we expect them to be rather imprecise.

Cache partitioning is often the suggested way to improve the predictability of multi-core hard real-time systems. This is largely due to the fact that partitioning the cache helps prevent inter-task contention for the same cache line, thus simplifying the computation of the worst-case execution time (WCET) of a task. However, even outside the realm of real-time systems, cache partitioning is used to improve the predictability and the overall throughput of systems [2], [9], [10].

We are aware of two lines of prior work on real-time scheduling with cache partitioning:

1) Per-core cache partitioning [3], [4]: Here, the shared cache is partitioned among the cores of a multi-core, and tasks are assigned to cores depending on their cache footprint.
2) Per-task cache partitioning [11]: Here, each task is allocated a certain share of the cache, and whenever a job of a task is scheduled, its share of the cache is made available to it privately.

Per-task cache partitioning is more flexible and may thus provide better performance. However, the prior work by Guan et al. [11] considers the cache space allocated to each task to be an input to their non-preemptive scheduling algorithm. In other words, the amount of cache space allocated to each task is *not* optimized to globally improve schedulability. Our aim is to fill this gap, by developing a novel method for computing offline schedules of sets of non-preemptable tasks using per-job cache partitioning. We initially limit the problem to sets of non-recurring tasks, but plan to extend it to recurrent task sets in the future.

## III. Problem Statement

### A. Machine Model

We consider a machine with $c$ identical cores and a shared cache of size $M$. The shared cache consists of $m$ *cache slices*, each of size $M/m$. At runtime, the set of cache slices may be partitioned arbitrarily among the cores.

Given a set-associative cache and way-based partitioning, the cache slices would correspond to the ways of the cache. Given a set-associative cache and set-based partitioning, which can be implemented in software, the cache slices would correspond to subsets of the cache's sets.

### B. Task Model

We consider a system which must schedule a set of $n$ real-time jobs, $B = \{J_1, J_2, \ldots, J_n\}$ on a machine as described in Section III-A. Each job $J_i$ is characterized by a 3-tuple $(C_i, a_i, d_i)$, where $C_i$ is the job's worst-case execution time, $a_i$ is the arrival time of job $J_i$, and $d_i$ is the (absolute) deadline before which the job must be completed. As the execution time depends on the amount of allocated cache space, $C_i$ is a function from $\{0, \ldots, m-1\}$ to $\mathbb{N}$, capturing the dependence of the job's execution time on the number of cache slices.

Given a particular schedule, the lateness of a job $J_i$ is defined as $L_i = f_i - d_i$, where $f_i$ denotes the finishing time of job $J_i$ under the given schedule.

### C. Scheduling Model

We consider global non-preemptive scheduling, where each job is assigned a fixed number of cache slices throughout its execution. Our goal is to compute an offline schedule that considers the available number of cores and cache slices and in which each job meets its deadline. For an optimal schedule, we would like to minimize the maximum lateness across all jobs.

## IV. Proposed Approach and Preliminary Results

### A. Partition Size Sensitivity

We used our low level timing analysis tool, `LLVMTA` [12] to analyze the worst-case execution times of jobs. For our evaluation, we used all the tasks in TACLeBench [13] and seven tasks generated from models using the SCADE tool [14] assuming fully-associative instruction and data caches ranging from 16 bytes to 16 KiB.

Figure 1 shows the results of this analysis for a subset of the tasks in TACLeBench[1]. Both the instruction and data caches were varied in lockstep for the analysis. All results are normalized to the WCET bound obtained for a cache of size 16 bytes. We observe that the tasks have differing timing behavior as the amount of cache available to them changes. Some smaller tasks benefit from additional cache space only at smaller cache sizes, and once their entire working set fits into the cache, there is no further improvement in the execution times. On the other hand, other tasks show only minor or no improvements in execution time until a significant part of their working set fits in the larger sized caches. Hence, we can see from Figure 1 that there is a benefit to partitioning the cache based on the actual requirements of each executing task.

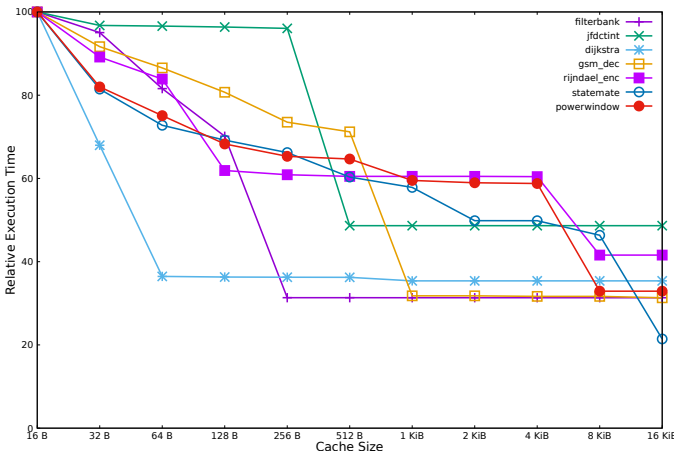



Fig. 1. Partition size sensitivity of selected TACLeBench tasks

### B. State Space Exploration

We model the scheduling problem as a path-finding problem in a directed acyclic graph. In such a graph, each edge represents a potential scheduling decision and each node represents the state of the systems after all the previous decisions have been carried out. Thus, each complete path in the graph represents a potential schedule of all the tasks. Such a graph will contain a large number of states and possible schedules, making an explicit search through all the possible paths infeasible for all but the smallest job sets.

We propose to use a variant of the A* algorithm [15], [16] to efficiently explore this graph. Using a heuristic function $h$, A* is steered towards promising schedules. Furthermore, if the heuristic function is *admissible*, the algorithm may safely discard large parts of the graph without sacrificing optimality. The characteristics of a heuristic cost function and the admissibility criteria are described in Section IV-C.

Following existing conventions, our algorithm uses a priority queue called OPEN and a list called CLOSED. The OPEN queue contains the set of states that have not yet been examined in increasing order of their heuristic cost. The CLOSED list is the set of states that have already been examined. We define *goal states* as those in which there remain no jobs that are yet to be scheduled. The state-space exploration algorithm then functions as follows:

1) Put initial state $\Phi$ in the OPEN queue.
2) Remove the lowest ranked element, $s$, which corresponds to the smallest heuristic cost, $h(s)$ and add it to CLOSED.
3) If based on the heuristic function, $h$, $s$ is guaranteed to not lead to a feasible schedule, then the job set is deemed infeasible and the algorithm stops[2]; else go to next step.

[1]Due to space constraints, we can show only a subset of the experimental results here. The full set of results can be found at: http://embedded.cs.uni-saarland.de/publications/ECRTS2018WIPExtended.pdf.

[2]As, following from A*, all other states in OPEN are then also guaranteed not to lead to a feasible schedule.

4) If $s$ is a *goal state*, then, according to A*, an optimal schedule has been found and the algorithm stops; else go to next step.
5) Expand the state $s$ by exhaustively matching all the ready jobs to available processors for each possible partition size from the available cache space. Each such matching produces a new state, $s'$. Compute the heuristic cost for each of the new states and add them to the OPEN queue. Then, go to step (2).

### C. Heuristic Cost Function

In this section we discuss the properties of a good heuristic cost function and our ideas for devising one. Since the heuristic cost function must be computed for a large number of states, it is imperative that it is cheap to compute.

An *admissible* heuristic cost function is one in which $h(s)$ always underestimates the value of the optimization goal for all schedules that can be constructed from $s$. The A* algorithm is guaranteed to find an optimal solution if the heuristic cost function is admissible. Hence, our heuristic should be admissible.

The heuristic cost function must also be "end-to-end", as the aim is to minimize the goal across the entire path, and not only for the remaining schedule that is yet to be explored. This is important, since otherwise, the algorithm would degenerate into a depth-first search.

Since the heuristic's job is to guide the algorithm towards the optimal schedule, the closer it estimates the maximum lateness of all completions of a partial schedule, the better the algorithm will perform.

We have two ideas to approximate the maximum lateness of all completions of a partial schedule from below:

1) By disregarding competition for cache space, assuming each remaining task is allocated the entire cache space.
2) By assuming that each task may be scheduled at its "resource-optimal" configuration, i.e. the one that minimizes the time-space product $C(h) \cdot h$.

### D. State Space Pruning

The original A* algorithm was designed to work with arbitrary graphs, i.e., with no knowledge of their structure. However, in our case, we can use the knowledge of what each state represents in order to prune certain paths which are guaranteed not to produce optimal results. We propose to extend the A* algorithm with an additional step as was previously suggested by Kwok et al. [17] to prune states which cannot possibly lead to an optimal schedule. The following types of states may be pruned:

1) **States that are dominated by another**: If it can be shown that one state is dominated by another state, we can safely prune it, since it cannot lead to an optimal solution. If $S_1$ and $S_2$ are two states such that every job scheduled in $S_2$, has either finished or is scheduled to finish earlier in $S_1$, then $S_1$ is said to dominate $S_2$. The efficient identification of dominated states is an important goal of future work.
2) **States which are identical**: When expanding a state $s$, multiple new states may be generated by matching a ready task to *isomorphic processors* [17]. These are sets of processors which are in a state such that assigning a task to either of them will produce equivalent schedules.

## V. Envisioned Solution

In this paper, we introduce our idea for offline non-preemptive multi-core scheduling with job-aware cache partitioning. We propose to generate schedules using a variant of the A* algorithm. Based on the preliminary results of the partition-size sensitivity of execution times, we are confident that this approach will yield significant improvements in the response times of jobs and in overall schedulability. Coming up with good heuristic cost functions and state-space pruning techniques is the subject of ongoing work. We also plan to extend our approach to recurrent task sets.

## References


[1] Certification Authorities Software Team (CAST), "Position Paper CAST-32A Multi-core Processors," November 2016.
[2] A. M. Molnos, S. D. Cotofana, M. J. M. Heijligers, and J. T. J. van Eijndhoven, "Throughput optimization via cache partitioning for embedded multiprocessors," in *ICSAMOS*. IEEE, 2006, pp. 185–192.
[3] S. Plazar, P. Lokuciejewski, and P. Marwedel, "WCET-aware Software Based Cache Partitioning for Multi-Task Real-Time Systems," in *WCET*, ser. OASICS, vol. 10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009, pp. 1–11.
[4] B. D. Bui, M. Caccamo, L. Sha, and J. Martinez, "Impact of cache partitioning on multi-tasking real time embedded systems," in *RTCSA*. IEEE Computer Society, 2008, pp. 101–110.
[5] J. H. Anderson, J. M. Calandrino, and U. C. Devi, "Real-time scheduling on multicore platforms," in *IEEE Real Time Technology and Applications Symposium*. IEEE Computer Society, 2006, pp. 179–190.
[6] Y. Ding and W. Zhang, "Multicore real-time scheduling to reduce inter-thread cache interferences," *JCSE*, vol. 7, no. 1, pp. 67–80, 2013.
[7] W. Zheng, H. Wu, and C. Nie, "Integrating task scheduling and cache locking for multicore real-time embedded systems," in *LCTES*. ACM, 2017, pp. 71–80.
[8] J. Xiao, S. Altmeyer, and A. D. Pimentel, "Schedulability analysis of non-preemptive real-time scheduling for multicore processors with shared caches," in *RTSS*. IEEE Computer Society, 2017, pp. 199–208.
[9] M. Ferdman, A. Adileh, Y. O. Koçberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *ASPLOS*. ACM, 2012, pp. 37–48.
[10] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *ISCA*. ACM, 2015, pp. 450–462.
[11] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-Aware Scheduling and Analysis for Multicores," *Emsoft'09*, p. 245, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1629335.1629369
[12] S. Hahn, M. Jacobs, and J. Reineke, "Enabling compositionality for multicore timing analysis," in *RTNS*. ACM, 2016, pp. 299–308.
[13] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sorensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *WCET*, ser. OASICS, vol. 55. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 2:1–2:10.
[14] "SCADE Suite." [Online]. Available: http://www.esterel-technologies.com/products/scade-suite/
[15] V. Kumar and L. N. Kanal, "A general branch and bound formulation for understanding and synthesizing and/or tree search procedures," *Artif. Intell.*, vol. 21, no. 1-2, pp. 179–198, 1983.
[16] J. Pearl and J. H. Kim, "Studies in semi-admissible heuristics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 4, no. 4, pp. 392–399, 1982.
[17] Y. Kwok and I. Ahmad, "On multiprocessor task scheduling using efficient state space search approaches," *J. Parallel Distrib. Comput.*, vol. 65, no. 12, pp. 1515–1532, 2005.

# Towards Synchronization in Prosa

Antonin Riffard
MPI-SWS
Kaiserslautern, Germany

Felipe Cerqueira
MPI-SWS
Kaiserslautern, Germany

Björn B. Brandenburg
MPI-SWS
Kaiserslautern, Germany

***Abstract*—We report on our ongoing work on the formalization of real-time resource sharing in Prosa, a framework based on the Coq proof assistant for the development of machine-checked schedulability analysis. We review the goals of the project, present a preliminary specification of a generic resource model, and then discuss the next steps and planned verification goals.**

## I. Introduction

Certification standards call for the temporal requirements of safety-critical real-time systems to be thoroughly checked and validated. For this purpose, the real-time systems community has proposed a wealth of schedulability analysis techniques. However, the complexity inherent in state-of-the-art analyses has made it increasingly difficult to verify such analyses "by hand," and in fact there are ample recent examples that suggest that human error is a significant cause for concern [1, 2, 3, 4, 5, 6, 7]. This unfortunately raises a fundamental question: why should certification processes admit schedulability analysis results as evidence of temporal correctness if the underlying analysis methods may not be entirely sound?

To address this question and provide a *trustworthy* analytical foundation for the real-time systems of tomorrow, the Prosa project [8] seeks to develop rigorous, provably-correct schedulability analyses. Specifically, Prosa is an open-source framework based on the Coq proof assistant that provides a general and extensible *formal specification* of real-time scheduling theory and allows users to develop *machine-checked* schedulability analyses that are guaranteed to be correct.

To date, Prosa has been successfully used to reason about various aspects of real-time scheduling theory, including uniprocessor and multiprocessor scheduling, response-time analyses [8], arbitrary processor affinity (APA) scheduling [1, 2], sustainability in the context of self-suspending tasks [9], and work on the certification of analysis tools by Guo et al. [10].

Nevertheless, despite these promising initial results, Prosa still has limited applicability in the analysis of real-world systems since it offers no support for mutual exclusion. To address this issue, we are currently working towards support for the verification of real-time synchronization protocols as well as their associated blocking analyses in Prosa. In this paper, we first discuss the goals of this extension and specific challenges, then present our preliminary model of critical sections and mutual exclusion, and finally conclude with a summary of the next steps and planned verification goals.

This work was funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — 391919384.

## II. An Initial Model of Mutual Exclusion

### A. Goals and Challenges

The main goal of this project is to extend the current Prosa specification with a generic resource model. Initially, we will only target mutual exclusion, but plan to include additional types of relaxed concurrency control mechanisms in the future, such as reader-writer locks and $k$-exclusion protocols.

To formalize this resource model, we must define resources and critical sections, restrict the notion of a valid schedule to enforce mutual exclusion, and also adapt the properties of work conservation and priority compliance to be coherent with the resource model.

Those changes must be applied to both uniprocessor and multiprocessor scheduling. In case of the latter, we must additionally consider that jobs are allowed to spin (*i.e.*, busy-wait) when a resource is not available. Thus, when defining this new schedule state, we must be careful to maintain compatibility with existing definitions.

Moreover, to match the assumptions of existing analyses, we must define the various types of nesting of critical sections. For example, a restrictive, but also common, assumption is that a job can hold at most one resource at a time (*i.e.*, non-nested critical sections). Another variation is to only allow well-nested critical sections (*i.e.*, given two critical sections A and B, either A is completely inside B or they do not overlap). An even more general model, such as found in the Linux kernel, allows critical sections to be nested arbitrarily, as long as requests are ordered to prevent deadlocks. In Prosa, we seek a single model for critical sections that is able to express all types of nesting.

### B. Prosa Foundations

Since Prosa already provides a foundation of common real-time scheduling concepts, we can focus directly on formalizing the resource model. To better understand the required changes, we begin with an overview of key definitions in Prosa.

The most basic notion in Prosa is that of jobs, which are represented by an opaque type called *Job*. Each *Job* is associated with certain parameters, modeled as functions, such as its actual execution cost:

```
Variable job_cost: Job → progress.
```

The above syntax expresses that each *Job* has a parameter called *job_cost* that maps a job to a value of type progress, which represents the amount of processor service required by a given job. The type progress, in turn, is just an alias of nat (the natural numbers), our model of discrete time.

Based on the notion of jobs, PROSA defines a uniprocessor schedule as a function that maps each time instant optionally to a job, represented by the type option *Job* (*i.e.*, either None or Some *j* with $j \in Job$), meaning that at any time, either the processor is idle or exactly one job is scheduled. The type time, in turn, is also an alias of nat.

```
Definition schedule := time → option Job.
```

Next, to model the progress and completion of execution, we define the central notion of service received by a job. Given a schedule *sched*, instantaneous and cumulative service of a job *j* are defined in PROSA as follows.

```
Definition service_at (j: Job) (t: time) : progress :=
  if sched t == Some j then 1 else 0.

Definition service (j: Job) (t: time) : progress :=
  \sum_(t' < t) service_at j t'.
```

When moving to the multiprocessor case, the only major change is the definition of schedule. Given the set of processors processor *num_cpus*, represented by integers in $[0, num\_cpus)$, PROSA defines a multiprocessor schedule as follows.

```
Definition schedule :=
  processor num_cpus → time → option Job.
```

Correspondingly, the definition of instantaneous service must also consider multiple processors. The definition of cumulative service, on the other hand, remains unchanged.

```
Definition service_at (j: Job) (t: time) : progress :=
    \sum_(cpu < num_cpus | sched cpu t == Some j) 1.
```

### C. Formalization of Resource Sharing

Having presented an overview of the basic concepts in PROSA, we now introduce critical sections, add constraints to the schedule to respect mutual exclusion, and finally integrate the task and resource models.

*1) Introducing Critical Sections:* During execution, a job may be required to access certain shared resources in isolation. Each of those execution intervals that are subject to mutual exclusion is called a critical section.

We model a critical section *cs* as an execution interval $[cs\_start, cs\_end)$ that requires some resource *cs_resource*. More precisely, critical sections and resources are represented by opaque types *CriticalSection* and *Resource*, with parameters *cs_start*, *cs_end* (the bounds of the interval) and *cs_resource* (the targeted resource). In addition, for any job *j*, we ensure that $0 \leq cs\_start < cs\_end \leq job\_cost\ j$.

```
Variable cs: CriticalSection.
Variable cs_start: CriticalSection → progress.
Variable cs_end: CriticalSection → progress.
Variable cs_resource: CriticalSection → Resource.
```

Note that *cs_start* and *cs_end* are expressed in terms of progress, the service received by the job up to a given time.

Next, we associate with each *Job* a list of critical sections.

```
Variable job_critical_sections: Job → list CriticalSection.
```

Since jobs can have multiple critical sections, each with arbitrary start and end times, our model is very permissive and allows any type of nesting. Nevertheless, it can also accommodate more restrictive constraints. For instance, non-nested critical sections can be specified as follows.

```
Definition no_overlapping_critical_sections :=
  ∀ j, ∀ cs1 cs2,
    cs1 ∈ job_critical_sections j →
    cs2 ∈ job_critical_sections j →
    critical_sections_overlap cs1 cs2 → cs1 = cs2.
```

This predicate states that, if two critical sections *cs1* and *cs2* intersect, they must be the same. To test for intersection, we use the predicate critical_sections_overlap, which compares the boundaries of the critical sections.

*2) Scheduling and Resource Allocation:* Having defined critical sections, we now specify the conditions under which a schedule is valid assuming a given resource model.

First, we must identify critical sections in the schedule. We say that job *j* has entered a critical section *cs* by time *t* iff at an earlier time $t' \leq t$, **(a)** job *j* has received enough service to enter the left boundary *cs_start* of the critical section, and **(b)** job *j* is *effectively executing* at time *t'*.

```
Definition job_has_entered_section
                    (j: Job) (cs: CriticalSection) (t: time) :=
  ∃ t', t' ≤ t ∧
      service sched j t' ≥ cs_start cs ∧
      service_at sched j t' > 0.
```

Next, we say that job *j* has exited the critical section iff its received service is no longer within the right boundary *cs_end*.

```
Definition job_has_exited_section
                    (j: Job) (cs: CriticalSection) (t: time) :=
  service sched j t ≥ cs_end cs.
```

Using the predicates above, we define whether job *j* has entered but not yet exited a critical section at time *t*.

```
Definition job_in_section
                    (j: Job) (cs: CriticalSection) (t: time) :=
  job_has_entered_section j cs t ∧
  ¬ job_has_exited_section j cs t.
```

Considering that multiple critical sections can refer to the same resource, we also define whether job *j* is holding a resource *r* at time *t*.

```
Definition job_holds_resource
                    (j: Job) (r: Resource) (t: time) :=
  ∃ cs, cs ∈ job_critical_sections j ∧
        cs_resource cs = r ∧ job_in_section j cs t.
```

Finally, we formalize the key property of mutual exclusion, *i.e.*, at any time *t*, at most one job can access each resource.

```
Definition enforces_mutual_exclusion :=
  ∀ r t, ∀ (j1 j2: Job),
    job_holds_resource j1 r t →
    job_holds_resource j2 r t → j1 = j2.
```

Note that when formalizing reader-writer or $k$-exclusion synchronization in the future, this definition will no longer be valid, but it can be easily generalized to such cases.

*3) Incorporating Spinning:* Recall that in uniprocessor systems, jobs must suspend when waiting for a resource. In multiprocessors, however, it is sometimes more desirable to spin (*i.e.*, busy-wait) while waiting for a resource that is going be released shortly.

Since a job that spins does not make progress in terms of service, incorporating spinning requires changing the representation of a schedule as follows.

```
Inductive cpu_state :=
  Idle | Running of Job | Spinning of Job.
Definition schedule :=
  processor num_cpus → time → cpu_state.
```

In the definition above, the keyword `Inductive` indicates that cpu_state is an enumerated type with three possible values:

- Idle, when the processor has no job scheduled (this corresponds to None in the old definition);
- Running $j$, when the processor is executing a job $j \in Job$ (this corresponds to Some $j$ in the old definition);
- Spinning $j$, when the currently scheduled job $j \in Job$ is busy-waiting for a resource.

Using the new processor state, we adapt the definition of service_at to count service only from Running processors:

```
Definition service_at (j: Job) (t: time) : progress :=
  \sum_(cpu < num_cpus | sched cpu t == Running j) 1.
```

Since the other definitions are built on top of service_at, no other changes are required and the property valid_resource_allocation remains valid.

*4) Incorporating Resources into the Task Model:* The remaining step is to incorporate job critical sections into the task model, so that we can later formalize blocking analyses.

For each task we define the maximum number and maximum length of critical sections (*task_num_cs* and *task_length_cs*, respectively). Then, we define two predicates to enforce such constraints for each individual job and critical section.

```
Variable task_num_cs: Task → Resource → nat.
Variable task_length_cs: Task → Resource → progress.

Definition num_critical_sections_is_bounded :=
  ∀ (j: Job), ∀ (r: Resource),
    count_mem r (map cs_resource (job_critical_sections j))
    ≤ task_num_cs (job_task j) r.

Definition critical_section_length_is_bounded :=
  ∀ (j: Job), ∀ cs, cs ∈ job_critical_sections j →
    cs_length cs
    ≤ task_length_cs (job_task j) (cs_resource cs).
```

In the above definition, the operation count_mem $r$ (map ...) counts the number of sections of job $j$ that access resource $r$.

## III. Future work

As future work, we plan to extend the specification by defining other types of nesting and formalizing reader-writer and $k$-exclusion synchronization. In addition, we aim to define real-time synchronization protocols and important concepts such as the notion of priority inversion.

After concluding the specification, we seek to verify existing blocking analyses, so that they can be integrated with schedulability analysis frameworks in Prosa.

As a first step, we will focus on the uniprocessor case with the *stack resource policy* (SRP) and verify the existing blocking bounds. Next, we will consider an extension of the protocol to multiprocessor platforms and formalize results for spin-based protocols, namely the *multiprocessor SRP* (MSRP) [11]. After verifying the analysis developed by Gai et al. [11] for non-nested critical sections, we will focus on the improved bound based on linear programming by Wieder and Brandenburg [12]. Ultimately, we plan to formalize the blocking analysis for well-ordered nested critical sections by Biondi et al. [13], and if possible, generalize this result by incrementally removing or weakening nesting hypotheses, using Coq's abilities to detect and flag required changes in the proofs.

## References


[1] A. Gujarati, F. Cerqueira, and B. Brandenburg, "Revised Version: Schedulability analysis of the Linux push and pull scheduler with arbitrary processor affinities, revision 1," Available at: https://www.mpi-sws.org/~bbb/papers/, 2015.

[2] ——, "Correspondence article: A correction of the reduction-based schedulability analysis for APA scheduling. To appear." *Real-Time Systems*, 2018.

[3] R. Bril, J. Lukkien, R. Davis, and A. Burns, "Message response time analysis for ideal controller area network (CAN) refuted," *Proceedings of the 5th International Workshop on Real-Time Networks (RTN'06)*, 2006.

[4] G. Nelissen, J. Fonseca, G. Raravi, and V. Nelis, "Timing analysis of fixed priority self-suspending sporadic tasks," in *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS'15)*, 2015.

[5] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, and D. de Niz, "Many suspensions, many problems: A review of self-suspending tasks in real-time systems," Department of Computer Science, TU Dortmund, Tech. Rep. 854, 2016.

[6] R. Davis, A. Burns, J. Marinho, V. Nelis, S. Petters, and M. Bertogna, "Global fixed priority scheduling with deferred pre-emption revisited," Univ. of York, Tech. Rep. YCS-2013-483, 2013.

[7] R. Devillers and J. Goossens, "Liu and Layland's schedulability test revisited," *Information Processing Letters*, vol. 73, no. 5, pp. 157–161, 2000.

[8] F. Cerqueira, F. Stutz, and B. Brandenburg, "PROSA: A case for readable mechanized schedulability analysis," in *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS'16)*, 2016.

[9] F. Cerqueira, G. Nelissen, and B. Brandenburg, "On strong and weak sustainability, with an application to self-suspending real-time tasks," in *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS'18)*, 2018.

[10] X. Guo, S. Quinton, P. Fradet, and J.-F. Monin, "Work In Progress: Toward a Coq-certified Tool for the Schedulability Analysis of Tasks with Offsets," in *Work in Progress Session, Real-Time Systems Symposium (RTSS'17)*, Paris, France, 2017.

[11] P. Gai, G. Lipari, and M. Di Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in *Proceedings of the 22nd Real-Time Systems Symposium (RTSS'01)*, 2001.

[12] A. Wieder and B. Brandenburg, "On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks," in *Proceedings of the 34th Real-Time Systems Symposium (RTSS'13)*, 2013.

[13] A. Biondi, B. Brandenburg, and A. Wieder, "A blocking bound for nested FIFO spin locks," in *Proceedings of the 37th Real-Time Systems Symposium (RTSS'16)*, 2016.

# Towards System-Wide Timing Analysis of Real-Time–Capable Operating Systems

Simon Schuster, Peter Wägemann, Peter Ulbrich, Wolfgang Schröder-Preikschat
Department of Computer Science, Distributed Systems and Operating Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

***Abstract*—In the context of static timing analysis of real-time operating systems, the usage of generically implemented algorithms necessitates annotation languages to express application-specific knowledge. That is, developers have to provide precise loop bounds or exclude program paths if automatic timing analysis fails or yields too pessimistic results. Current annotation approaches are not able to express and propagate information across all layers of complete real-time systems (i.e., application and system-call layer, operating system, machine-code level).**

**To solve this problem, we present our work in progress within the SWAN project to enable System-wide WCET Analyses. Specifically, we provide details on PLATINA, a high-level annotation language, which is processed by an optimization-aware compiler and timing-analysis infrastructure. PLATINA enables expressing parametric program-flow facts based on the real-time system's context-sensitive state (e.g., number of currently active tasks), which are propagated through and usable on all layers of the real-time system. Eventually, PLATINA allows determining if a generically implemented system is real-time–capable and whether timing bounds can be guaranteed during execution.**

## I. INTRODUCTION & MOTIVATION

The worst-case response time (WCRT) is a vital temporal property of tasks with hard deadlines. Its evaluation demands a sound worst-case execution time (WCET) analysis of all implementation artifacts. Accordingly, static timing analysis of real-time applications is a well-established field with commercially available tools (e.g., aiT [1]) that yield precise bounds on the WCET in practice. They, nonetheless, rely on a largely static program structure, which is inherent to safety-critical applications, to accurately infer data and control flows, to, for instance, bound loops and resolve function pointers.

However, tasks are typically embedded and executed in a broader system's context using a real-time operating system (RTOS). Consequently, its implementation has to be subject to the same temporal analysis. The induced overhead is typically treated as constant and pessimistically added in a deferred analysis step to each task's WCRT [2], [3]. The reason is that system calls and preemptive scheduling intermit data and control flows and thus necessitate a dedicated analysis of kernel paths. The use of such a compositional approach is consequently only feasible when given a static setting, which means that operating system (OS) implementations are tailored to a specific set of statically decidable application parameters.

A relatively new development in safety-critical real-time systems is the employment of dynamic runtime environments, such as Real-Time Linux [4], [5] that correspond more to a general-purpose OS and promote code reuse by generically implemented algorithms and interfaces. A reliable indicator of this development is the future Adaptive AUTOSAR standard [6], which addresses the increasing complexity of driver-assistance and autopilot functions in vehicles. Here, the aggregation of the individual (i.e., application and kernel) WCETs is fraught with inevitable overestimations that rise tremendously with system complexity [7]. The cause is in the loss of a tailored and static OS implementation and the decoupling of control flows within the application and the OS kernel resulting from the system's dynamic system-call layer and runtime reconfigurability. Consequently, system facilities are designed to serve all possible application scenarios and thus exhibit dynamic data structures and execution paths, which in turn jeopardize conventional analyses [3]. However, flexible deployment of dynamic RTOS in safety-critical settings still requires realistic bounds for the induced overheads.

*Our Contribution:* We present our work in progress on system-wide WCET analysis that makes available context-specific knowledge to the kernel analysis by PLATINA, an expressive annotation language. PLATINA features parametric flow facts that can be linked to system state (e.g., number of currently active tasks) in an application- and context-sensitive manner. In conjunction with an optimization-aware compiler and timing-analysis infrastructure, we can infer tighter yet sound bounds for generically implemented real-time systems.

## II. RELATED WORK & PROBLEM STATEMENT

Colin and Puaut [8] were among the first to pinpoint the fundamental problems of OS WCET analysis: meaningful construction of global control-flow graphs over kernel boundaries. They identified indirect function calls, immanent in the system calls interface, as well as dependencies between application properties and kernel loop bounds as crucial issues. Their approach was to modify the system's source code, restoring a statically analyzable implementation. Despite these tedious and non-generalizable measures, a high degree of overestimation (avg. 86 %) remained. Later Sandell et al. [9] reported a large number of "uninteresting" kernel paths (e.g., error handling) in their analysis. Furthermore, they observed the mediocre performance of the data-flow analysis within the OS kernel. They used an annotation language with constant expressions at assembly level to address the problems, which, however, involved a high degree of recurring effort with still unsatisfactory reduction of pessimism.

*Acknowledgments:* This work is supported by the German Research Foundation (DFG), in part by Research Grant no. SCHR 603/9-2, the SFB/Transregio 89 "Invasive Computing" (Project C1), and the Bavarian Ministry of State for Economics under grant no. 0704/883 25.

Among others, Schneider [7] determined the dynamic reconfigurability and system-call interface, the tracing of call graphs, as well as the internal feedback between the RTOS and the applications as further challenges towards realistic WCET estimations. He proposed an integrated WCET and scheduling analysis as a potential solution. In 2009 Lv et al. compiled a survey [3] on RTOS-analysis attempts, from which they derived a set of challenges. Like Schneider, they question the usefulness of single, global WCET estimates for individual operations but instead suggest a parametric analysis that captures specific WCETs for each invocation.

Since then, research focused on circumventing the problem by tailoring the OS to be deterministic again. Undecidable artifacts (e.g., loop iterations, indirect calls) are eliminated by application-specific source-code modifications: For example, in the seL4 kernel [2] preemption points in loops are added to limit the length of consecutive kernel execution. In our work on SysWCET [10], we leveraged the scheduling semantics to eliminate globally infeasible system execution paths by using a tailored operating system. However, for larger RTOSs (e.g., Real-Time Linux) these code-tailoring approaches are not feasible, due to the high complexity and recurring effort.

*Our challenge:* Instead of tailoring, we opt for a generic annotation of the OS implementation and subsequent context-aware WCET analysis that proves real-time capabilities specifically for a given application setting. We identified three practical symptoms on the fundamental problem of context-sensitive control flows: (1) Control-flow reconstruction issues, (2) paths that are either unanalyzable or become infeasible in certain contexts, and (3) overly pessimistic bounds as context knowledge cannot be automatically inferred. The resulting challenge is therefore to provide a unified way of formulating, passing, and evaluating state- and context-sensitive flow facts in a system-wide WCET analysis, especially when analysing OS operations. As assembly-level annotations [2], [9] are not an option due to their poor reusability and manageability, this requires parametric annotations at the source-code level.

## III. Approach

We tackle our challenge by SWAN, an approach to enable System-wide WCET Analyses. Figure 1 illustrates its fundamental concept: the core (middle) is PLATINA, a parametric annotation language expressing context-sensitive information on the source-code level. The annotations are associated with system facts (top) that hold context-dependent information. Finally, to achieve a context-aware WCET analysis (bottom), SWAN provides a semantic-preserving transformation from code to assembly level. We detail those steps in the following.

Our goal with PLATINA is to bridge the immanent semantic gap between application and kernel analysis by propagating context-dependent information (*system facts*) to the low-level WCET analysis in an automated fashion. Hence, we provide parametric (i.e., context-dependent) annotations to address said three challenging symptoms that so far prevent tighter bounds on generic OS operations: (1) Annotation of indirect function calls to aid the control-flow reconstruction. This is a key

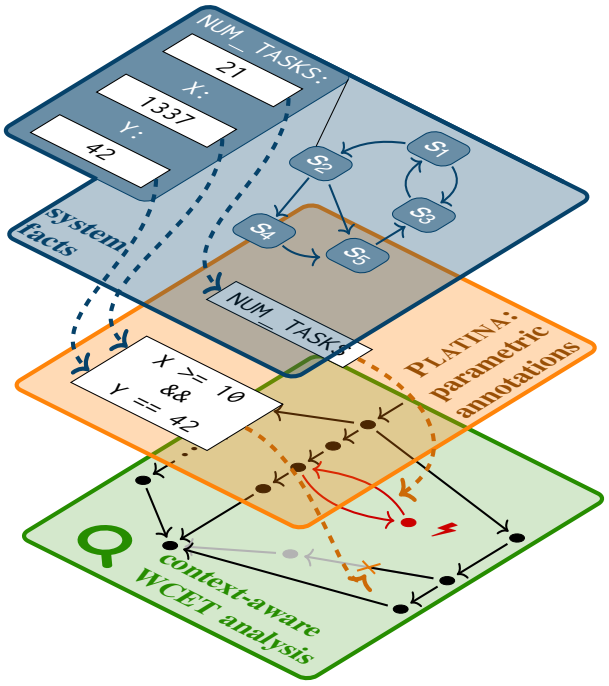



Fig. 1. Information propagation within SWAN: semantic system facts are gathered from a semantic model. Parametric annotations use these pieces of information to express the semantics of context-sensitive execution flow at the source code level. After lowering to the assembly-code level, the static WCET analysis leverages this information to exclude infeasible execution flows.

enabler for the subsequent propagation of context-dependent flow facts. (2) State-dependent annotation of branches in conditional executions to eliminate paths that become infeasible only in certain contexts. (3) Context-sensitive annotations to bound computations by application and configuration knowledge. These building blocks form an expressive annotation language that allows associating the semantics of individual system facts with the actual execution flow. To reflect the complexity of control flows within the OS, PLATINA's annotation language allows the programmer to combine multiple system facts and even reuse expressions by defining custom facts or functions. Furthermore, to foster long-term maintainability of annotations, we decided to integrate them at the annotated program point within the source code (e.g., C++ program).

The *system-facts layer* parametrizes and instantiates PLATINA annotations. Therefore, the layer provides and stores static and application-specific configuration knowledge as well as context-dependent information, which we call *system facts*: the atomic entities on which the annotation language operates. Higher-level analysis, such as our SysWCET [10] approach to incorporate scheduling semantics, also engage at this level to further derive and enrich the model with further system facts.

Finally, the annotation expressions are evaluated over a given context of system-fact values within our *context-aware WCET analysis*, which is based on and extends the PLATIN toolkit [11]. While boosting (re-)usability, our earlier decision to keep the annotations at source-code level poses the question of how they can be propagated soundly to the assembly-code level of the WCET analysis. Here, we rely on optimization-

aware compilation [12] to transform context-specific flow information into the machine-code–level control-flow graph. Consequently, our lowering preserves the annotation semantics, which ultimately allows us to infer specific *facts* on the execution *flow* from the contexts of system calls. Examples include loop bounds that are specific to the number of runnable tasks or paths that are only infeasible in the given context. That way, we obtain context-sensitive and thus more accurate bounds for OS overheads, which proofs that the OS is real-time capable in the given setting. Our lowering, while based on an optimization-aware compilation, enables parametrization without recompilation: the same binary can be evaluated for arbitrary system facts and in multiple contexts, which retains scalability and usability even for large implementations.

In summary, SWAN provides sound propagation of high-level, semantic, context-sensitive information throughout the compilation and analysis processes down to the level of the static WCET analysis. The parametric annotation mechanism allows obtaining tailored, context-aware timing bounds even for generic OS kernels. Thus, SWAN proves their real-time capability in a specific context with potentially higher analysis accuracy compared to traditional decoupled WCET analysis.

## IV. Preliminary Results

To demonstrate the feasibility of SWAN, we conducted a case study on FreeRTOS and relevant parts of the Linux kernel (i.e., scheduler). In both cases, traditional WCET analysis does not obtain realistic bounds; partly it even fails in the reconstruction of the control-flow graph. This observation, first of all, substantiates the need for an integrated, context-sensitive approach to WCET analysis. Our first results with SWAN are promising, and we were already able to annotate and analyze various intricate parts of both settings. As quantitative results depend on the given application scenario, we graphed the impact of our approach on the example of FreeRTOS's `vSuspendTask` system call in Figure 2: Whenever a task is suspended, a reschedule is triggered, entailing the selection of the next runnable task. Here, FreeRTOS relies on a set of queues (one per priority level), which are probed with decreasing priority. For our experiments, we configured FreeRTOS for the ARM Cortex-M4 platform and a total of 42 priority levels. Without our extensions, PLATIN computed (supported by constant annotations) a static upper bound of 6485 cycles for the worst case, which assumes no runnable task in any of the queues and therefore requires 42 probing steps. However, as there is a linear relationship between the number of probing steps and the system call's WCET, this bound is too pessimistic in most cases: we observed divergence of over 178 %. With SWAN, we were able to accurately express this context dependency by combining the number of priority levels and the priority of the next task. The former is a system-configuration property; the latter is a context-dependent property (e.g., available from our state-transition graph [10]). Consequently, our bounds correctly reflect the maximum number of runnable tasks for any given state and context; thus ultimately relieving OS overheads from unnecessary pessimism.

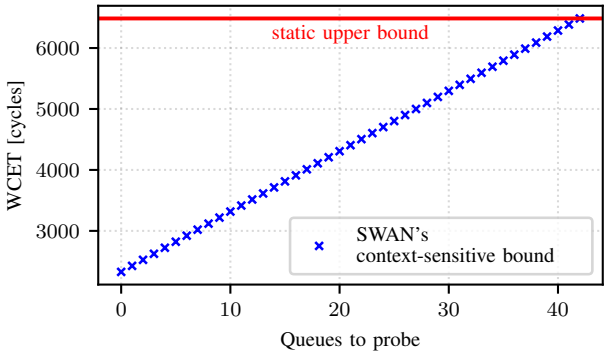



Fig. 2. Runtime of the `vTaskSuspend` system call in relation to the number of task queues (total set of 42) that are probed until a runnable task is found.

## V. Conclusion and Future Work

In this paper, we presented SWAN, our ongoing effort on context-sensitive WCET analysis of RTOSs. Its key element is PLATINA, a parametric annotation language to express dependencies between a system's control-flow and application states generically. Combined with system facts and an optimization-aware compiler and timing analysis, we can both prove real-time capabilities in a given context as well as eliminate overly pessimistic constant bounds on RTOS overheads.

We currently finish our prototype and case study of FreeRTOS and Linux. Beyond this proof of concept, we consider the sheer size of, for example, Real-Time Linux as another fundamental challenge. Thus, our next steps are devoted to improving traceability of system facts and annotations as well as the overall scalability and usability. In the future, we see great promise for the integration of high-level application and RTOS semantics to further reduce analysis pessimism.

## References


[1] AbsInt., "aiT worst-case execution time analyzers," absint.com/ait.

[2] B. Blackham, Y. Shi, S. Chattopadhyay, A. Roychoudhury, and G. Heiser, "Timing analysis of a protected operating system kernel," in *Proc. of RTSS '11*, 2011.

[3] M. Lv, N. Guan, Y. Zhang, Q. Deng, G. Yu, and J. Zhang, "A survey of WCET analysis of real-time operating systems," in *Proc. of ICESS '09*, 2009.

[4] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "LITMUS$^{RT}$: A testbed for empirically comparing real-time multiprocessor schedulers," in *Proc. of RTSS '06*, 2006.

[5] B. B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, UNC, 2011.

[6] AUTOSAR, "AUTOSAR the next generation – the adaptive platform," autosar.org/fileadmin/files/presentations/EUROFORUM_Elektronik-Systeme_im_Automobile_2016_-_FUERST_Simon.pdf, 2016.

[7] J. Schneider, "Why you can't analyze RTOSs without considering applications and vice versa," in *Proc. of WCET '02*, 2002.

[8] A. Colin and I. Puaut, "Worst-case execution time analysis of the RTEMS real-time operating system," in *Proc. of ECRTS '01*, 2001.

[9] D. Sandell, A. Ermedahl, J. Gustafsson, and B. Lisper, "Static timing analysis of real-time operating system code," in *Leveraging Applications of Formal Methods*, ser. Lecture Notes in Comp. Science, 2004.

[10] C. Dietrich, P. Wägemann, P. Ulbrich, and D. Lohmann, "SysWCET: Whole-System Response-Time Analysis for Fixed-Priority Real-Time Systems," in *Proc. of RTAS'17*, 2017.

[11] P. Puschner, D. Prokesch, B. Huber, J. Knoop, S. Hepp, and G. Gebhard, "The T-CREST approach of compiler and WCET-analysis integration," in *Proc. of SEUS '13*, 2013.

[12] B. Huber, D. Prokesch, and P. Puschner, "Combined WCET Analysis of Bitcode and Machine Code Using Control-flow Relation Graphs," in *Proc. of LCTES '13*, 2013.

# FF-DBF-WIN: On the Forced-Forward Demand-Bound Function Analysis for Wireless Industrial Networks

Miguel Gutiérrez-Gaitán and Patrick Meumeu Yomsi
*CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal*
*Email: {mjggt, pamyo}@isep.ipp.pt*

***Abstract*—Wireless Industrial Networks (WINs) have brought to the forefront the need for real-time strategies to ensure network schedulability. The Demand Bound Function (DBF) has recently been borrowed from the multicore scheduling theory and adapted to the wireless industrial domain to compute the network demand. However, a more precise estimation can be obtained by using alternative supply/demand analyses. This paper proposes the forced-forward demand bound function to estimate the network demand and better determine the schedulability of WINs.**

## 1. Application domain

Wireless Industrial Networks (WINs) have become one of the enabling technologies of Industry 4.0. Advantages such as flexibility, easy deployment and low cost devices have led to its gradual incorporation into smart factories, intelligent manufacturing systems, among other industrial contexts. Nevertheless, from a communication and network viewpoint, WINs still pose significant technical challenges compared to traditional wireless sensor networks (WSNs). WirelessHART, ISA100.11a, WIA-PA and IEEE 802.15.4e are a few examples of standards developed during the last two decades to specifically address some of these industrial requirements. Here, techniques such as interference minimization, redundancy, frequency hopping and power efficiency are combined together with recommendations in the standards to satisfy reliability and real-time requirements. Along the same line, with the advent of recent Cyber-Physical Systems and Internet of Things, protocols such as 6LowPAN (IPv6 over Low-Power Wireless Personal Area Networks), RPL (Routing Protocol for Low-Power and Lossy Networks), CoAP (Constrained Application Protocol) and 6TiSCH (IP over the TSCH mode of IEEE 802.15.4e) [1] have focused their standardization efforts on aspects such as the integration of smart devices into the Internet.

Despite all the attempts so far and the entire body of knowledge available in the literature (see [1] for a comprehensive survey) to address common wireless industrial requirements, important hurdles such as real-time schedulability analysis, i.e., the ability of each flow to meet all its timing requirements, have received very little attention in the context of WINs.

## 2. Motivation

Willig et al. [2] recognize both reliability and timing guarantees as two of the most desirable characteristics of WINs. Because wireless channels are usually prone to transmission errors, a cornerstone on the way to guarantee these features for such a system is the schedulability of the network flows. To this end purpose, channel contention and transmission conflicts are the two common delays that affect the end-to-end transmission of each network flow and thus its schedulability. Recently, Xia et. al [3] have studied these two components and factored the contribution of each in the end-to-end transmission delay of each flow by borrowing analysis techniques from the multicore scheduling theory. Specifically, the authors focused on the relationship between the network supply and demand of the flows in any time interval. However, the proposed analysis was based on the "Demand Bound Function" (DBF) [4] concept to determine the schedulability of the flow set, which does not consider all flows that can potentially contribute to the network demand, unfortunately.

In the literature on multicore scheduling theory, the forced-forward demand bound function (FF-DBF) [5] offers a tighter alternative to estimate the workload demand of a system as it includes potential contributions that are left aside by DBF.

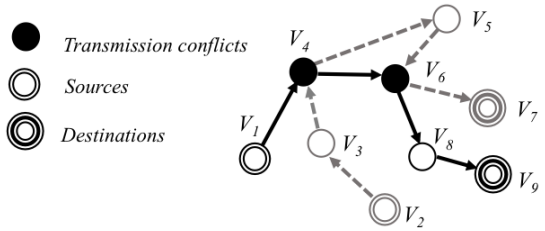
## 3. Problem statement

We consider the model of execution proposed in [3] where a wireless industrial network is represented as a graph $G = (V, E, m)$. Here, $V$ denotes the set of network devices (or nodes), $E$ represents the set of edges between the nodes and $m$ is the number of channels. For implementation purpose, we assume that a number $x \geq 2$ of nodes can communicate if they are not more than $d$ meters apart in an area $A$ such that the following equation holds true:

$$\frac{x}{A} = \frac{2\pi}{d^2\sqrt{27}} \tag{1}$$

Given this setting, we consider a set of $n$ network flows $F \stackrel{\text{def}}{=} \{F_1, F_2, \ldots, F_n\}$ to be transmitted from their respective source to their respective destination by following an Earliest Deadline First (EDF) scheduling algorithm [6]. Each flow $F_i$, with $i \in [1, n]$, is modeled by using a periodic end-to-end communication scheme between its source and its destination through a 4-tuple $\langle C_i, D_i, T_i, \phi_i \rangle$, where $C_i$ is the number of hops between source and destination; $D_i$ is the relative deadline; $T_i$ is the period; and $\phi_i$ is the routing path of the flow. These parameters are given with the interpretation that: each flow releases a potentially infinite number of instances

(or jobs). The $k^{\text{th}}$ job (with $k \geq 1$) of flow $F_i$ is denoted as $F_{i,k}$ and is released at time $r_{i,k}$ such that $r_{i,k+1} - r_{i,k} \stackrel{\text{def}}{=} T_i$. Job $F_{i,k}$ has to be completely transmitted to its destination node by its absolute deadline, i.e., $d_{i,k} \stackrel{\text{def}}{=} r_{i,k} + D_i$. We assume that $D_i \leq T_i$, i.e., only a single job of flow $F_i$ is being/can be transmitted at any time instant. The number of hops between the source and the destination of flow $F_i$, denoted as $C_i$, represents its transmission time when it does not suffer any external interference whatsoever from the other flows. The last parameter, $\phi_i$, represents the actual route of all jobs issued from flow $F_i$. Figure 1 depicts an example of a WIN with two flows $F_1$ and $F_2$ together with their associated parameters according to this model. In this figure, flow $F_1$ is transmitted from node $V_1$ to $V_9$ via nodes $V_4$, $V_6$ and $V_8$; and flow $F_2$ is transmitted from node $V_2$ to $V_7$ via nodes $V_3$, $V_4$, $V_5$ and $V_6$. Note that there are transmission conflicts at nodes $V_4$ and $V_6$.




| | $C_i$ | $D_i$ | $T_i$ | $\varphi_i$ |
|---|---|---|---|---|
| $F_1$ —— | 4 | 15 | 20 | $V_1$, **$V_4$**, **$V_6$**, $V_8$, $V_9$ |
| $F_2$ ---- | 5 | 20 | 30 | $V_2$, $V_3$, **$V_4$**, $V_5$, **$V_6$**, $V_7$ |

Figure 1. An example of a wireless industrial network.

The objective is to provide an accurate network supply/demand analysis by using the forced-forward demand bound function, in order to guarantee the schedulability of all the flows in the system. A simulation experiment will be carried out to compare the performance for both the DBF and FF-DBF-based analyses. Finally, a rigorous per flow analysis will be conducted and the gain in terms of accuracy of the analysis will be evaluated.

## 4. Observations and preliminary results

As mentioned in Section 2, channel contention and transmission conflicts are the two common delays affecting the end-to-end transmission time of network flows in WINs.

1) **Channel contention:** refers to the delay produced by a high priority job occupying all channels at a time instant.
2) **Transmission conflict:** refers to a job transmission delayed by the transmission of a higher priority job.

Xia et al. [3] factored these two delays in their proposed supply/demand bound analysis method.
▷ About Point 1), the authors assume that the flows are executed on a multi-processor platform and each channel of the WIN is mapped as one processor. They bound the supply bound function (sbf)[1] of an industrial network as follows.

$$\text{sbf}(0) = 0 \ \wedge \ \forall \ell, k \geq 0 : \text{sbf}(\ell + k) - \text{sbf}(\ell) \leq \text{Ch} \times k \quad (2)$$

where Ch is the number of channels in the network and they derive the network demand caused by channel contention in any time interval of length $\ell$ as

$$\text{DBF}(\ell)^{\text{Ch}} = \frac{1}{m} \sum_{i=1}^{n} \max \left\{ \left( \left\lfloor \frac{\ell - D_i}{T_i} \right\rfloor + 1 \right) \cdot C_i, 0 \right\} \quad (3)$$

▷ About Point 2), the authors provided an estimation of the network demand caused by the transmission conflicts under the EDF scheduling algorithm by tuning the result proposed by Saifullah et al. in [7]. Here, the transmission of two flows are in conflict when their paths overlap, i.e., if we consider two flows $F_i$ and $F_j$ such that $F_i$ has a higher priority than $F_j$, then at a given node, say $V_x$, the progress of the jobs generated from $F_i$ cannot be delayed by jobs generated from $F_j$, but jobs generated from $F_j$ may be delayed by jobs generated from $F_i$ at node $V_x$ and at all subsequent nodes shared by the transmission paths of $F_i$ and $F_j$. Based on this observation, Xia et al. [3] estimated the network demand caused by the transmission conflicts as follows

$$\sum_{i,j=1}^{n} \left( \Delta(ij) \max \left\{ \left\lceil \frac{\ell}{T_i} \right\rceil, \left\lceil \frac{\ell}{T_j} \right\rceil \right\} \right) \quad (4)$$

In Equation 4, $\Delta(ij) \stackrel{\text{def}}{=} \sum_{k=1}^{\delta(ij)} Len_k(ij) - \sum_{k'=1}^{\delta'(ij)} (Len_{k'}(ij) - 3)$. Here, $Len_k(ij)$ denotes the length of the $k^{\text{th}}$ path overlap and $\delta(ij)$ is the number of path overlaps; and $Len_{k'}(ij)$ and $\delta'(ij)$ refer to the delay caused by path overlaps with length as least 4. Finally, Xia et al. computed an upper bound on the network demand in any time interval of length $\ell$ by summing Equations 3 and 4. They concluded on the schedulability of the flow set $F$ by checking if Equation 5 is satisfied.

$$\sum_{F_i \in F} \text{DBF}(F_i, \ell) \leq \text{sbf}(\ell), \ \forall \ell \geq 0 \quad (5)$$

Since Equation 3 is using the DBF function to estimate the network demand, the contributions of some jobs arriving and/or having deadlines outside the interval of interest are not taken into account, unfortunately. This would lead to an underestimation of the network demand, which in turn may result in a less accurate supply/demand bound analysis. In order to circumvent this issue, we propose to borrow the FF-DBF function[2] from the multi-core scheduling theory, instead. This function is defined for a single flow $F_i$ as follows.

$$\text{FF-DBF}(F_i, \ell) \stackrel{\text{def}}{=} q_i \cdot C_i + \begin{cases} C_i & \text{if} \quad \gamma_i \geq D_i \\ C_i - (D_i - \gamma_i) & \text{if} \quad D_i > \gamma_i \geq D_i - C_i \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

In Equation 6, $q_i \stackrel{\text{def}}{=} \left\lfloor \frac{\ell}{T_i} \right\rfloor$ and $\gamma_i = \ell \mod T_i$. Figure 2 illustrates a comparison between the demand evaluated by using FF-DBF and the demand evaluated by using the classical DBF in an interval of length $\ell$ for an arbitrary task (or flow).

1. The supply bound function - sbf($\ell$) - of a network is the minimal transmission capacity provided within a time interval of length $\ell$.

2. The FF-DBF refines the DBF and allows us to include the potential missing contributions into the cumulative computational demand.

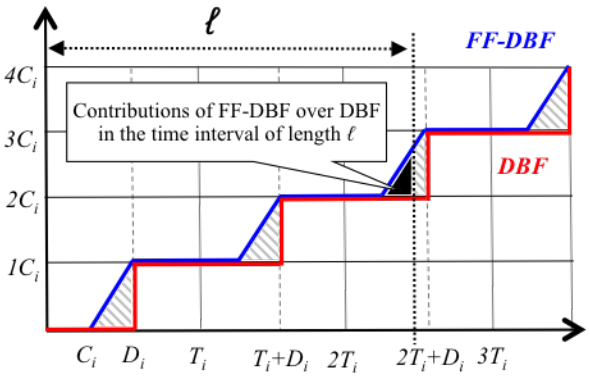



Figure 2. Pictorial representation of FF-DBF vs. DBF.

Given the model defined in Section 3, we adapt the FF-DBF function to the wireless industrial networks domain by revisiting Equations 3 and 4 as follows.

▷ *About Equation 3:* We recall that this equation provides us with the network demand caused by channel contention in any time interval of length $\ell$. By using the FF-DBF function instead to this end, that estimation is given by:

$$\text{FF-DBF}(\ell)^{\text{Ch}} = \frac{1}{m}\sum_{i=1}^{n}\text{FF-DBF}(F_i,\ell) \tag{7}$$

▷ *About Equation 4:* This equation allowed us to estimate the network demand caused by the transmission conflicts. Since these conflicts are agnostic to the network topology, there is no need to alterate this contribution, therefore this factor remains the same as previously.

As a result of these two observations, a more precise upper-bound of network demand is obtained by using the FF-DBF($\ell$) function defined as follows:

**FF-DBF-WIN.** A forced-forward demand bound function FF-DBF($\ell$) of a given WIN in any time interval of length $\ell$ is defined by summing Equations 4 and 7. Formally,

$$\begin{aligned}\text{FF-DBF}(\ell) = &\frac{1}{m}\sum_{i=1}^{n}\text{FF-DBF}(F_i,\ell)+\\ &\sum_{i,j=1}^{n}\Delta(ij)\max\left\{\left\lceil\frac{\ell}{T_i}\right\rceil,\left\lceil\frac{\ell}{T_j}\right\rceil\right\}\end{aligned} \tag{8}$$

A pseudo-code for the computation of this function is presented in Algorithm 1.

**Algorithm 1** FF-DBF Algorithm for WINs

**Input:** $m; F_i(C_i, D_i, T_i, \o_i); \ell; \Delta(ij); n;$
**Output:** FF-DBF($\ell$)
*Initialisation* : FF-DBF($\ell$) $\leftarrow 0; q_i \leftarrow 0; \gamma_i \leftarrow 0;$
1: **for** $i = 1$ to $n$ **do**
2: $q_i \leftarrow \left\lfloor \frac{\ell}{T_i} \right\rfloor;$
3: $\gamma_i \leftarrow \ell \mod T_i;$
4: **if** $(\gamma_i \geq D_i)$ **then**
5: FF-DBF($\ell$) += $q_i \times C_i + C_i;$
6: **else**
7: **if** $(\gamma_i \geq D_i - C_i)$ **then**
8: FF-DBF($\ell$) += $q_i \times C_i + C_i - (D_i - \gamma_i);$
9: **else**
10: FF-DBF($\ell$) += $q_i \times C_i;$
11: **end if**;
12: **end if**
13: **end for**
14: FF-DBF($\ell$) $\leftarrow \frac{1}{m} \times$ FF-DBF($\ell$);
15: **for** $i = 1$ to $n$ **do**
16: aux $\leftarrow \max\left\{\left\lceil\frac{\ell}{T_i}\right\rceil, \left\lceil\frac{\ell}{T_j}\right\rceil\right\};$
17: FF-DBF($\ell$) += $\Delta(ij) \times$ aux;
18: **end for**
19: **return** FF-DBF($\ell$)

## 5. Envisioned Solution and Conclusion

In this ongoing research, we proposed the forced-forward demand bound function (FF-DBF) as a refinement of the demand bound function (DBF) to charactize the network demand in wireless industrial networks. We believe that Equation 8 is more accurate for the estimation of an upper bound on the network demand as it allows us to take into account potential missing contributions, left aside by the classical DBF function, into the cumulative computational demand in any time interval (see Figure 2). Also, we are confident that it will outperform the analysis proposed by Xia et al. [3] for the schedulability of periodic flows in WINs and will lead us to a more accurate supply/demand bound analysis. Now we seek to: ($i$) formally demonstrate this claim; ($ii$) conduct simulation experiments to compare the performances for both DBF and FF-DBF-based analyses and ($iii$) thus validate the efficiency of the proposed approach. Finally, we will also derive a rigorous per flow analysis and evaluate the gain in terms of computational complexity.

## Acknowledgments

This work was partially supported by National Funds through FCT (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] D. V. Queiroz, M. S. Alencar, R. D. Gomes, I. E. Fonseca, and C. Benavente-Peces, "Survey and systematic mapping of industrial wireless sensor networks," *Journal of Network and Computer Applications*, 2017.

[2] A. Willig, K. Matheus, and A. Wolisz, "Wireless technology in industrial networks," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1130–1151, 2005.

[3] C. Xia, X. Jin, and P. Zeng, "Resource analysis for wireless industrial networks," in *Mobile Ad-Hoc and Sensor Networks (MSN), 2016 12th International Conference on*. IEEE, 2016, pp. 424–428.

[4] 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.

[5] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "Improved multiprocessor global schedulability analysis," *Real-Time Systems*, vol. 46, no. 1, pp. 3–24, 2010.

[6] 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.

[7] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-end communication delay analysis in industrial wireless networks," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1361–1374, 2015.

# On Multi-Level Preemption in Ethernet

Mubarak Adetunji Ojewale, Patrick Meumeu Yomsi and Geoffrey Nelissen
CISTER Research Centre, ISEP, Polytechnic Institute of Porto, Portugal
Email: {mkaoe, pamyo, grrpn}@isep.ipp.pt

***Abstract*—Ethernet is increasingly being considered as the solution to high bandwidth requirements in the next generation of timing critical applications that make their way in cars, planes or smart factories to mention a few examples. Until recently, ethernet frames used to be transmitted exclusively in a non-preemptive manner. That is, once a frame starts transmitting on a switch output port, its transmission cannot be interrupted by any other frame until completion. This constraint may cause time critical frames to be blocked for long periods of time because of the transmission of non-critical frames. The IEEE 802.3br standard addressed this issue by introducing a one-level ethernet frame preemption paradigm. In this approach, frames transmitted through a switch output port are classified as express frames or preemptable frames, depending on their priority levels. Express frames can preempt preemptable frames and two frames belonging to the same class cannot preempt each other. While this partially solves the problem for express frames, all preemptable frames can still suffer blocking irrespective of their priority level. In this work, we investigate the feasibility and advantages of multi-level preemptions in time-sensitive ethernet networks.**

## I. Introduction

Most systems today are made of several embedded devices interconnected through networks. Cars, planes, train and factories for instance contain tens to hundreds of sensors, actuators and computers that must communicate with timing guarantees. Real-time applications require responsiveness i.e., timely and correct reaction to events, which largely depends on the ability of data to move in a predictable manner on the network. Ethernet is the emerging communication technology in industrial and automotive domains. Its relatively cheaper price and its high bandwidth capacity make it the ideal replacement for previous communication infrastructures generally adopted in these domains. However, the legacy ethernet standard was mainly targeting non real-time applications and desirable capabilities like preemption, global time synchronisation across the network, frame duplication and re-transmission were initially missing. In order to provide system designers with these desirable features, several modifications have been made to the standards over the years. The IEEE 802.1p task group, for example, introduced a mechanism to specify a Class of Service (CoS) for ethernet frames in order to expedite the transmission of high priority frames [1]. The CoS of a frame signifies its priority and the frames are transmitted according to their CoS, highest priority first. Other features like time-triggered transmission, global clock synchronisation, credit based shaping, among others, have also been added to the ethernet to make it more suitable for real-time applications [2].

One modification made to the ethernet to support real-time communication is reported in the IEEE 802.3br and 802.1Qbu standards, which specify a frame preemption protocol for ethernet networks. Preemption implies that a frame that has already started its transmission on a switch output can be suspended in order for a more "urgent frame" to be transmitted through the same port. The transmission of the preempted frame is resumed only after the urgent frame has been fully transmitted. Preemption allows a high priority frame with stringent timing requirements to be transmitted more promptly, but this is achieved at the cost of some overheads. Fig. 1 illustrates a scenario where a high priority frame is transmitted by preempting a low-priority one. Upon the occurrence of each preemption, the standards specify some additional information to be added to the preempted frames so as to notify the network devices about the preemption, thereby impacting the transmission link utilization.

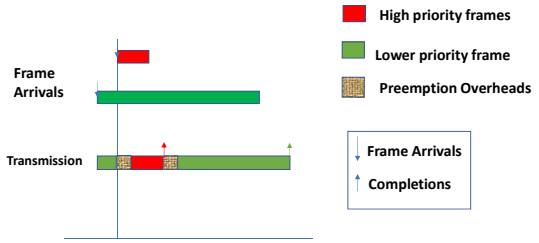



Fig. 1: Illustration of frame preemption.

Before the specification of the IEEE802.3br standard, ethernet frames used to be transmitted in a non-preemptive manner [3]. Any low-priority frame could block any high priority frame for long periods of time depending on the low-priority frame's size. It is to circumvent this limitation that frame preemption was defined in the IEEE 802.1Qbu [4]. This standard specifies one-level preemption for ethernet frames since only two MAC service interfaces are supported: a preemptable MAC (pMAC) interface and an express MAC (eMAC) interface. Frames assigned to the eMAC service interface are referred to as *express* frames and those assigned to the pMAC interface as *preemptable* frames.

A critical look at the one-level preemption, however, raises some concerns. With the current specification, only express frames are allowed to preempt preemptable frames and frames of the same class cannot preempt each other [2]. In typical real-time applications, there are traffic classes that are not classified as express but nevertheless have timing constraints and should not be blocked for long periods by lower priority frames. To illustrate this, consider a medium priority frame (black frame in Fig. 2). With one-level preemption, the

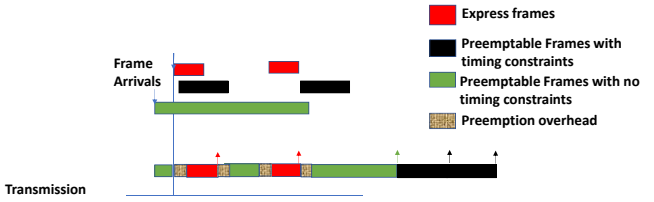



Fig. 2: Frame transmission under 1-level preemption.

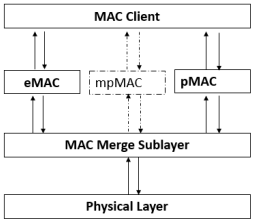



Fig. 3: The MAC merge sublayer managing service interfaces.

medium priority frame, which may be important to the smooth operation of a real-time application, can be blocked by a large lower priority frame (green frame) if both share the pMAC interface. This is because a preempted frame must complete its transmission before any other non-express frame can be transmitted. Consequently, this limitation has a negative effect on medium priority frames that are not uncommon in real-time applications. Note that if the medium priority frame was instead classified as express, then it could block more urgent frames, thereby defeating the whole purpose of introducing express frames in the first place.

Most work in the literature on this topic have been studying the effect of frame preemption on worst-case end-to-end transmission delays. The authors in [5] and [3], for example, showed that frame preemption reduces the transmission delays of express traffic significantly, but it has adverse effect on preemptable traffic. Thiele and Ernst [3] presented a Compositional Based Analysis (CPA) to provide guarantee on the end-to-end transmission delay of ethernet traffic with one-level preemption under Standard Ethernet and Time Sensitive Networking (TSN). To the best of our knowledge, no work has investigated the feasibility of multiple preemption levels on ethernet networks, especially for the scheduled traffic that are non-express, but with stringent timing constraints. In this work, we consider three levels of priorities and 2 levels of preemption to investigate the feasibility of multi-level preemption in ethernet.

## II. Problem Statement

We consider a network traffic consisting of $n$ streams $s_1, s_2, \ldots, s_n$ partitioned into traffic classes: *express traffic*, *medium priority preemptable traffic* (mpFrames) and *Best Effort preemptable traffic* (bpFrames). Stream $s_i$, with $i \in [1, n]$, consists of a potentially infinite number of frames $s_i^j$ $(j \geq 1)$ with an inter-arrival time of at least $T_i$ units between two consecutive frames. Frame $s_i^j$ is characterised by its arrival time $a_i^j$ and its size $c_i^j$. Express traffic frames have very strict timing requirements. Preemptable traffic frames are divided

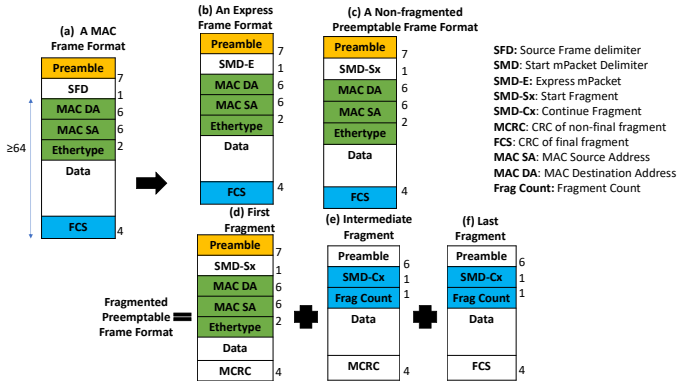



Fig. 4: Ethernet frame formats as specified in IEEE 802.3 Standards. Numbers represent sizes of each field (in bytes)

into two classes: mpFrames with strict timing constraints yet not as stringent as those of express traffic frames and the bpframes with no timing constaints at all. We assume that each stream is assigned a unique fixed priority and all frames generated from this stream inherit its priority. We also assume that any express frame has a higher priority than all preemptable frames and the following constraints are enforced:

- ▷ Any express frame can preempt all preemptable frames,
- ▷ Any mpFrame has higher priority than all bpFrames and therefore, can preempt these,
- ▷ Frames from the same class cannot preempt each other.

With the above assumptions and contraints, we investigate the feasibility of supporting multi-level preemption to limit the blocking of mpFrames by bpFrames.

## III. Preemption in Ethernet Networks

Preemption occurs at the MAC merge sublayer, which is between the physical and the MAC layers (See Fig. 3). Frames at this sublayer are called *mFrames*. The sublayer may preempt a preemptable mFrame currently being transmitted and may also prevent it from starting its transmission citeStandard802.3br-2016. Before each mFrame transmission, the sublayer verifies if the next switch/node supports preemption by performing a verification operation (see citeStandard802.3br-2016, page 42 for details). Preemption capability is enabled only after the verification operation confirms that it is supported. When this is the case, additional information are added to the mFrame headers, describing its preemption characteristics. In addition, it is important to preserve the ethernet frame format when mFrames are preempted. IEEE 802.3br ensures this by defining mFrame formats in a preemption enabled environment. Fig. 4 shows that express frames (see Fig. 4b) differ from normal MAC frames (see Fig. 4a) by only 1 octet, referred to as "Start Frame Delimiter" (SFD) by replacing the MAC frame SFD with "Start Mframe Delimiter-Express" (SMD-E) in the frame format. In practice, the SFD and SMD-E have the same value. Similarly, a preemptable frame that is not preempted (see Fig. 4c) differs from a normal MAC frame only in that the SFD is replaced with "Start MFrame Delimiter Start Fragment" (SMD-Sx). When a frame is preempted, the

first fragment of the frame differs from a non preempted preemptable frame only in that the error checking code (FCS) of the fragment is replaced with a newly generated mFrame error checking code (mCRC) by the MAC merge sublayer (see Fig. 4d). All other fragment headers only contain a preamble, "Start Mframe Delimiter for Continuation fragment" (SMD-Cx) and frag_count (see Fig. 4e) to track subsequent fragments. The last fragment ends with the FCS of the original preempted frame (see Fig. 4f).

At the receiving node, a Medium Independed Interface (xMII) inspects the SMD for each frame upon arrival. The value of the SMD indicates whether the received frame is express or preemptable [1]. Express frames (containing SMD-E) are processed by an Express Filter and preemptable frames by a "Receive processing" construct. Receive processing ensures that fragments of a preempted frame are received completely and in correct order using the mCRC and the frag_count. The mCRC is computed such that all fragments of a preempted frame end with the same mCRC, except the last one which ends with the original frame FCS. Frag_count is used to monitor the correct order of frame arrivals and to detect missing frames. A mismatch in the mCRC after a sequence of arrival of fragments indicates the end of the reception of the preempted frame, i.e., the last fragment has been received and the frame transmission is complete.

## IV. Feasibility of Multi-level Preemption and Recommendations

If an mFrame containing SMD-Sx (signalling the start of the transmission of a new preemptable frame) arrives at a node and Receive processing has not completed the reception of a previous preempted frame, Receive processing ensures that the MAC detects a "FrameCheckError" in the partially received frame (see [1], page 44). This mechanism implies that the node can detect the start of another preemptable frame, which is important to support multi-level preemption. Although the start of another preemptable frame would be flagged as an error, the IEEE 802.3br standard states that other techniques may be employed to respond to an incomplete frame transmission as long as the MAC behaves as though a FrameCheckError occurred. This submission opens the door to multi-level preemption specification, while still conforming to the standard. To this end, we recommend the specification of a mechanism to ensure the transmission of a frame in a higher preemption class without jeopardizing the integrity of the preempted frame. This operation should be performed such that the receiver node/switch correctly resumes the reception of the preempted frame later on.

The standards do not describe any mechanism to reassemble more than one frame in a buffer. We recommend the specification of such a mechanism to enable multi-level preemption as the buffer must be able to correctly reassemble and transmit a second frame, while already containing fragments of a first frame. In addition, the xMII that separates express frames from preemptable frames can be configured to distinguish between different priority levels for preemptable frames. As such, no additional frame filtering mechanism would be required for multi-level preemption.

We believe that the current preemptable frames format in the standards [6] is sufficient to handle multi-level preemption. To this end, we recommend that new values be defined for the one octet SMD contained in the header (See Fig. 4) to support more preemption levels. The standard currently defines eleven values for this octet. Additional values can be defined to check the level of preemption supported by the next node and to indicate the frame preemption levels.

We believe that a switch node supporting multi-level preemption can interoperate with those supporting only one-level or no preemption at all. With the new recommended SMD values, the MAC merge sublayer will be able to verify if the next node supports preemption and if this is the case, how many levels are supported. If just one level of preemption is supported, then all preemptable frames are transmitted on a single pMAC interface and multi-level preemption is disabled. In this case, all non-express frames are treated as preemptable frames and will not preempt each other. In the case preemption is not supported at all, frames are transmitted as already specified in the IEEE 802.1Q standards.

## V. Expected Results and Conclusion

At this stage of this Work-in-Progress, we examined the feasibility of multi-level preemption in ethernet networks and provided a set of recommendations. Now, we seek to develop a formal worst case transmission delay analysis of frames assuming multi-level preemption and conduct experiments to demonstrate its effectiveness in time sensitive ethernet networks. An improvement is expected for medium priority frames with affordable preemption overhead in terms of buffer size and SMD definitions.

## Acknowledgment

This work was partially supported by National Funds through FCT (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] "IEEE standard for local and metropolitan area networks–bridges and bridged networks," *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pp. 1–1832, Dec 2014.

[2] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, "Ultra-low latency (ULL) networks: A comprehensive survey covering the IEEE TSN standard and related ULL research," *CoRR*, vol. abs/1803.07673, 2018.

[3] D. Thiele and R. Ernst, "Formal worst-case performance analysis of time-sensitive ethernet with frame preemption," in *2016 IEEE 21st ETFA*, Sept 2016, pp. 1–9.

[4] "IEEE approved draft standard for local and metropolitan area networks-media access control (MAC) bridges and virtual bridged local area networks amendment: Frame preemption." *P802.1Qbu/D3.1, September 2015*, pp. 1–50, Jan 2015.

[5] W. K. Jia, G. H. Liu, and Y. C. Chen, "Performance evaluation of ieee 802.1qbu: Experimental and simulation results," in *38th Annual IEEE Conference on Local Computer Networks*, Oct 2013, pp. 659–662.

[6] "IEEE standard for ethernet amendment 5: Specification and management parameters for interspersing express traffic," *IEEE Std 802.3br-2016 (Amendment to IEEE Std 802.3-2015)*, pp. 1–58, Oct 2016.

# Task Mapping in a Regularity-based Resource Partitioning Hierarchical Real-Time System

Guangli Dai* Pavan Kumar Paluri* Albert M. K. Cheng
Department of Computer Science, University of Houston, Houston, TX 77004, USA
email: {gdai, pvpaluri, amcheng}@uh.edu

***Abstract*—Virtualization via a Hierarchical Real-Time System (HiRTS) is gaining popularity in recent years. This paper introduces a Centralized Task-Partition Architecture to connect the two levels of an HiRTS whose resource level adopts the Regularity-based Resource Partition (RRP) Model. Based on this architecture, we propose two models. The Regular-Partition Periodic Task Mapping (RPM) Model aims at mapping periodic tasks to regular partitions online. The Regular-Partition Compositional Task Mapping (RCM), on the contrary, considers the mapping of both periodic and sporadic tasks to regular partitions. As the first attempt to map tasks to partitions based on RRP model, this paper presents how RPM and RCM are modeled with ideas from online Multiple Knapsack Problem (MKP). Thus, this paper completes the full picture of a RRP-based HiRTS.**

## I. Introduction

Virtualization is being increasingly applied in embedded systems as it can enhance the reliability [1], improve flexibility [2] and handle thermal exposures of MultiProcessor System-on-Chip (MPSoC) at the architecture level [3]. In a two-level Hierarchical Real Time System (HiRTS) [4], there are two levels: the resource level and the task level. This paper proposes an architecture and two models for connecting the resource and task levels while maintaining the transparency between them to enhance the flexibility of the model. This paper adopts Regularity-based Resource Partition (RRP) Model in the resource level. Since RRP model has been decently studied these years[5, 6, 7], the specifically designed bridge between the task and resource levels introduced in this paper, shall complete the whole picture of the RRP-based HiRTS.

A few models have been proposed for the resource level: the Periodic Model [8], the Bounded-Delay Model [9], the Explicit Deadline Periodic (EDP) Model [10] and the Regularity-based Resource Partition (RRP) Model [5]. Amongst all the aforementioned models, RRP can utilize the resources with less deviation between the ideal and the actual resource allocation [7]. Hence, this paper adopts RRP for the resource level.

The RRP Model divides physical resources into different **partitions**. As shown in the example in Fig. 1, a Virtual Machine (VM) may contain multiple partitions, e.g., VM1 contains partitions 1 and 2 [2]. Besides, a task on a certain VM is mapped only to a partition contained by this VM, e.g., Task 1 and 2 are mapped to Partition 1. Several studies have been done on the RRP Model, e.g., [5] and [6], which theoretically introduce how partitions in RRP are defined by the parameters **Availability Factor** and **Supply Regularity**. Better scheduling algorithms based on RRP are also proposed for single core [6] and multi-core scenarios [11], which greatly increase the utilization of the physical resources. Li and Cheng [7] also prove that uniprocessor scheduling algorithms can be directly reused for task scheduling on a single partition.

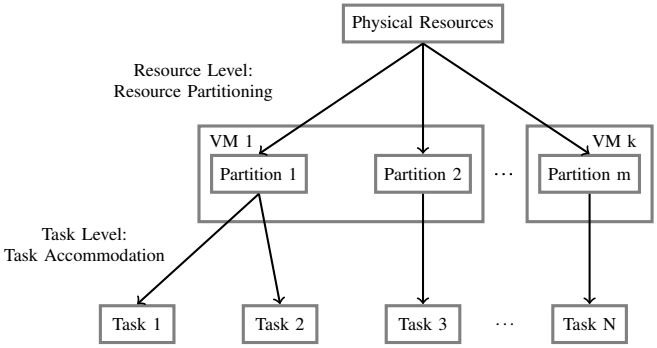



Fig. 1. A two-level resource partitioning framework.

However, as stated before, a virtual machine may contain several partitions, each with different computational power. How to schedule tasks on different partitions has received little attention. This paper proposes a centralized task-partition architecture connecting the resource level and the task level based on the RRP Model. Usage of Shared Memory segment (SM) to map tasks to partitions, also enables transparency between the two levels by maintaining code level independence.

Based on this architecture, we correlate the model of mapping periodic tasks to regular partitions to the model of online Multiple Knapsack Problem (MKP), which further enables us to develop the Regular-Partition Periodic Task Mapping (RPM) Model. Considering that RPM only takes periodic tasks into count, which is not quite realistic, we further propose the Regular-Partition Compositional Task Mapping (RCM) Model to map both sporadic and periodic tasks to regular partitions.

Being the first ever attempt to map a task to a partition based on RRP Model, this paper borrows certain ideas from different variants of online MKP [12]. Though the work is still under progress, We have already proved that a simple strategy Best Fit can reach a constant competitive ratio of $\frac{1}{2}$ for RPM under certain type of inputs. This paper introduces the architecture

*The first author and the second author have made equal contributions.

†Supported in part by the US National Science Foundation under Award No. 1219082.

in Section II and the two models, together with the evaluation of them in Section III.

## II. Centralized Task-Partition Architecture

The bridge between the resource level and the task level becomes increasingly vital as there is a pressing need for partitions in resource level to accommodate the incoming task requests. In this section, we first introduce how the allocation of tasks to partitions are done. Then we discuss the method of implementing the architecture.

### A. Task Manager (TM)

TM is a platform where tasks are uploaded and mapped as shown in Fig. 2. All tasks shall upload their information (Worst Case Execution times, Arrival times, Relative deadlines and etc.) in **upload()** to TM. TM then queues up all the requests in a large queue(s) and performs mapping (**map()**) on each available task request in the queue one by one. Discussion concerning the size of the queue will be presented in the future work. The function map() maps the incoming task to a suitable partition. It also updates the partition information when the previous task is mapped to a partition so that the current task request is well aware of the changes at the partition level. Post mapping, the task requests are dispatched (**dispatch()**) to their assigned partitions through shared memory segment model. Hence, after mapping, the task request shall have a new field appended to the request i.e., Partition ID along with the existing task information.

How the mapping is performed in map() is discussed in Section III. Next discuss the efficiency of the mechanism.

### B. The advantages of TM

Due to the real-time constraints, tasks are supposed to be mapped to a certain partition when they arrive. Adopting a TM reduces the number of communication channels needed for communication. Besides, since partitions are set when the system boots, the shared memory between TM and partitions can also be built in advance and initial information of partitions can be stored in TM at the very beginning. By doing so, the time taken to map a task to its assigned partition can be significantly reduced. Next, we discuss the efficiency of deploying the shared memory segment.

### C. Shared Memory Segment (SM)

The dispatching of task request to the assigned partition post mapping can thus be implemented either using the concept of pipes or Shared Memory (SM). However, pipe is limited to the communication between two processes with the same parent process. Hence, SM suits the scenarios where processes are independent from one another better. Also, as the message packet that has to be sent across is quite large in size, usage of SM can be beneficial and efficient at the same time. The dispatch() using SM shall be dealt in detail in the future work.

## III. Regular-Partition Task Mapping Models

In this section, we consider how to map periodic tasks and sporadic tasks to a set of regular partitions. To begin with, we briefly review the RRP Model first.

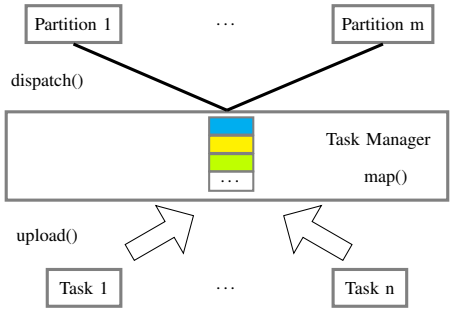



Fig. 2. Centralized Task-Partition Architecture .

### A. A Brief Review in RRP Model

The RRP Model divides physical resources into different partitions. Each partition $P_i$ is characterized by the availability factor $\alpha_i$ and the supply regularity $k_i$. $\alpha_i$ is the ratio of time units $P_i$ takes during a hyper period in a physical resource. Suppose a partition occupies a CPU core, then its availability factor would be 1. The availability factor is always less than or equal to 1 [6].

Let $S_i(t)$ denote the time units $P_i$ takes before time $t$. Then we can define the instant regularity [6] to be $Ir_i(t) = S(t) - t\alpha_i$. Accordingly, we can define the supply regularity $k_i$.

$$k_i = \lceil \max Ir_i(t) - \min Ir_i(t) \rceil \tag{1}$$

With Equation 1, we can see the supply regularity, which is an integer no smaller than 1, bounds the deviation between the ideal and the actual resource allocation. Depending on the value of $k_i$, $P_i$ can be a **regular partition** ($k_i = 1$) or an **irregular partition** ($k_i > 1$).

Here we present the characteristic that only regular partitions possess, with the following theorem.

*Theorem 1:* A regular partition $P_i$ can guarantee that it owns at least one time unit every $\frac{1}{\alpha_i}$ time units.

Theorem 1 indicates regular partitions can be well utilized and predicted with merely the availability factors and the supply regularities. Further details supporting theorem 1 shall be dealt in detail in future work. In next subsection, we will show how the characteristics shown in theorem 1 can be utilized to simplify the mapping problem.

### B. Regular-Partition Periodic Task Mapping (RPM)

In this subsection, we consider how to map periodic tasks to regular partitions. To accomplish the goal, we first show how to perform the schedulability test for a task set on a partition with merely partition's availability factor and supply regularity.

To judge whether mapping a periodic task to a regular partition is schedulable or not, having knowledge of the partition's availability factor and supply regularity is sufficient. We first define a periodic task set $T$, element $j$ in which $T_j = (c_j, d_j, p_j, R_j)$ where $c_j$ stands for Worst Case Execution Time (WCET), $d_j$ for the relative deadline, $p_j$ for the period and $R_j$ for arrival time of $T_j$. No information about $T_j$ shall be known before it arrives.

*Theorem 2:* A task set $T$ is schedulable on a regular partition $P_i$ iff

$$\sum_{j=1}^{|T|} \frac{c_j}{\min\{d_j, p_j\}} \leq \alpha_i \quad (2)$$

The Proof of Theorem 2 will be given in the future work. With Theorem 2, we can reduce the problem of mapping periodic tasks to regular partitions into an online MKP.

First, we review the definition of an online MKP problem. Each item is defined by its size, value and arrival time while each bin is defined by its capacity. The sum of the sizes of the items placed in a certain bin cannot exceed the capacity of the bin. When an item is successfully placed, we gain its value as a reward. The goal is to maximize the values gained [12].

Here, partition is a bin, whose capacity is its availability factor while a periodic task $T_j$ is an item with a size of $\frac{c_j}{\min\{d_j,p_j\}}$. As for the value of $T_j$, we have two different assumptions. If we assume each task to be equally important to the system, values obtained on putting any task into a bin remain the same, i.e., 1. With this assumption, the mapping problem is thus reduced to the **unit case** of online MKP [12]. On the other hand, if our primary goal is to maximize the utilization ratio of physical resources, we set the value of a task to be the same as its size $\frac{c_j}{\min\{d_j,p_j\}}$. Such an assumption corresponds to the **proportional case** of online MKP [12]. Besides, in this online mapping problem, without knowledge of the future, the system will not abandon a task actively unless there is no space for the task. This resembles the setting of the Fair Bin variant of online MKP [13].

This is the RPM model derived from online MKP. Algorithms for the two variants of the RPM Model and the analysis of these algorithms will be presented in the future work.

### C. Regular-Partition Compositional Task Mapping

In last subsection, the absence of sporadic tasks in the task set brings in decent simplification to the model. However, RPM is not realistic enough as sporadic tasks are important in real-time systems. In this subsection, we attempt to formulate a new model Regular-Partition Compositional Task Mapping (RCM), with both periodic and sporadic tasks considered.

Unlike periodic tasks, sporadic tasks are not that predictable. Therefore, we regard sporadic tasks as single-instance tasks in our model since it is not worthy to hold the resources for the next instance that may not come in the near future. Once an instance of the sporadic tasks is generated, it will be uploaded to the task manager and mapped to a partition in run time. This instance leaves the system when it is done without occupying any resources further.

Hence, we redefine the compositional task set $T$ so that both sporadic and periodic tasks can be accommodated into the model. Task $T_j$ in $T$ is defined by $T_j = (c_j, d_j, p_j, R_j, D_j)$. Similarly, $c_j$ is the WCET and $d_j$ is the relative deadline while $p_j$ is the period. $R_j$ is the arrival time and $D_j$ is the leaving time. To be specific, after $D_j$, no execution on $T_j$ is needed. For periodic tasks that do not leave the system, $D_j$ is $+\infty$. For sporadic tasks, regarded as single-instance tasks, they always have: $d_j = p_j$ and $D_j = R_j + d_j$.

Fortunately, the schedulability test shown in Equation 2 is still valid for the compositional task set. Therefore, model RCM is exactly the same as model RPM except that task $T_j$ will leave the system after $D_j$. RCM problem is more realistic yet harder to solve as well.

### D. Evaluation of RPM and RCM

Closer to the traditional online MKP, RPM is a problem easier to solve since it does not involve the leaving of tasks. So far, we already prove that a simple strategy Best Fit can reach a constant competitive ratio of $\frac{1}{2}$ in RPM.

Though RCM is harder to solve, the fact that it shares most of the features with RPM makes it less challenging. For instance, the reason why Best Fit performs well is that it gathers the spare space for the unknown future, which is also valid and a necessity in solving RCM. Based on these features, we develop another algorithm considering not only the capacity but also tasks' leaving time, while allocating tasks. More details and contributions will be given in the future work.

## REFERENCES


[1] Y. Laarouchi, Y. Deswarte, D. Powell, J. Arlat, and E. De Nadai, "Enhancing dependability in avionics using virtualization," in *Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems*. ACM, 2009, pp. 13–17.

[2] S. Xi, M. Xu, C. Lu, L. T. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in xen," in *Embedded Software (EMSOFT), 2014 International Conference on*, pp. 1–10.

[3] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Härtig, L. Hedrich *et al.*, "Design and architectures for dependable embedded systems," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on*, pp. 69–78.

[4] X. Feng, "Design of real-time virtual resource architecture for large-scale embedded systems," Ph.D. dissertation, 2004.

[5] A. K. Mok and X. Alex, "Towards compositionality in real-time resource partitioning based on regularity bounds," in *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*. IEEE, 2001, pp. 129–138.

[6] Y. Li and A. M. Cheng, "Static approximation algorithms for regularity-based resource partitioning," in *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pp. 137–148.

[7] Y. Li and A. M. K. Cheng, "Transparent real-time task scheduling on temporal resource partitions," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1646–1655, 2016.

[8] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE, 2003, pp. 2–13.

[9] A. K. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*. IEEE, 2001, pp. 75–84.

[10] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using edp resource models," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 129–138.

[11] Y. Li and A. M. Cheng, "Toward a practical regularity-based model: The impact of evenly distributed temporal resource partitions," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 4, p. 111, 2017.

[12] M. Cygan, Ł. Jeż, and J. Sgall, "Online knapsack revisited," *Theory of Computing Systems*, vol. 58, no. 1, pp. 153–190, 2016.

[13] J. Boyar, K. S. Larsen, and M. N. Nielsen, "The accommodating function: A generalization of the competitive ratio," *SIAM Journal on Computing*, vol. 31, no. 1, pp. 233–258, 2001.

# Implementing the Regularity-based Resource Partition Model on RT-Xen

Kevin Bailey*, Albert Cheng†, Pavan Kumar Paluri‡, Guangli Dai§ and Carlos Rincon¶
Real-Time Systems Laboratory
Department of Computer Science, University of Houston in Houston, TX, USA
{krbailey*, amcheng†, pvpaluri‡, gdai§, carincon¶}@uh.edu

***Abstract*—As technologies begin pushing the boundaries beyond what was previously thought unreachable, new algorithms in the space of hierarchical real-time systems (HiRTS) are becoming more necessary than ever. With hardware evolving at such a rapid pace, the current scheduling models are showing their age. The periodic model has been implemented most commonly due to its simplicity towards resource partitioning, but moving forward the regularity-based model has the potential to take its place. Using a widely accepted and open-source hypervisor, RT-Xen, we can ensure accurate performance of real multiprocessor machines. Here, we will discuss our current implementation of the MulZ partitioned multi-resource scheduling algorithm for regular partitions on the RT-Xen platform, as well as what is to come and the challenges that need to be tackled.**

## I. Introduction

Hierarchical Real-Time Systems (HiRTS) have seen a surge in popularity with the ability to assign multiple virtual CPUs to a single physical CPU. A commonly used hypervisor to do this is the open source project Xen [7], and one patch of it in particular called RT-Xen [4] is what we are most interested in. RT-Xen has given better results than Xen in terms of scheduling and accuracy using LITMUS$^{RT}$, as seen in [3]. However, with the introduction of this MulZ (Multiple Z-Sequences) partitioned multi-resource scheduling algorithm using the regularity-based resource partition (RRP) model, some issues with RT-Xen arise that we need to address.

RT-Xen 2.0 has been discussed in the past, but the newest version has not been covered as much. This newest version, RT-Xen 2.2, is based on Xen 4.6 and is compatible with later versions, while adding a few other features. Some of these include the ability to swap between Earliest Deadline First (EDF) and Rate-Monotonic (RM) schedulers on the fly, while also improving the efficiency of the implementations. Because of these changes, RT-Xen 2.2 renamed its global scheduler to Deferrable Server (rtds) and still maintains its original rt-partition scheduler. Although these changes are the latest to be implemented, our issues with these schedulers still remain.

While RT-Xen did improve on the default schedulers of Xen [3], one of the many issues that will inevitably be run into for partitioning algorithms is that of task migration across multiple CPUs. In its current form, RT-Xen only allows multiple VCPUs to be scheduled within an individual PCPU. These VCPUs can have masks to pin to different PCPUs, but during execution this can lead to many issues without something to manage the swapping safely. Now, if these tasks being scheduled on CPU-1 were in need of migration to another partition on a different CPU, they would be unable to do so natively. To remedy this issue, we have begun developing our own scheduler to use in the latest version of RT-Xen that will allow these migrations to happen, which will be necessary as we move closer to the regularity-based model.

Advancing towards an actual implementation of the MulZ algorithm, a few things should be discussed: the understanding of the Regular Partition and the terms associated with it are necessary, along with knowledge of Approximation Algorithms. A culmination of all the prior results from these topics will help with understanding an implementation of this MulZ algorithm and RRP model. The relevant terms will be discussed in the following sections, as well as what our objectives are to implement this on the RT-Xen hypervisor.

## II. Approach

To begin, we should describe the notion of a Supply Function, $S_P(t)$. The supply function of a resource partition represents the number of its available time-units from 0 to $t$. This is used in conjunction with the weight of a partition, $W_P = \frac{q}{p}$, where $q$ is the length of a time-unit sequence and $p$ is its period, to define the Instant Regularity, $I_P(t) = S_P(t) - t \cdot W_P$. So now, as shown in Li and Cheng [1] a resource partition P is considered regular if and only if

$$\forall t_1,\ t_2,\ |I_P(t_1) - I_P(t_2)| < 1$$

The MulZ algorithm can only be applied to these types of partitions since they are very close to evenly distributed. The downside of these regular partitions is that they are much more complex, leading to difficulties implementing them as opposed to periodic resource partitions. However, once implemented this MulZ algorithm looks very promising especially for multi-core scheduling problems [2].

In addition to these concepts, knowledge of Approximation Algorithms is also a requirement, specifically for regular partitions. Li and Cheng [2] have shown that Extended Approximating Boundary Sequences (E-ABS) achieve a lower approximation overhead than a normal ABS. The typical ABS/E-ABSes that we look at and are of interest for the MulZ algorithm are:

The work is supported in part by the United States National Science Foundation (NSF) under Award No. 1219082.

$$\mathcal{G}_{n,m} = \{\frac{1}{nm}, \frac{1}{nm^2}, \frac{1}{nm^3}, ...\}$$
$$\mathcal{H}_{n,m} = \{\frac{n-1}{n}, \frac{n-2}{n}, ..., \frac{1}{n}, \frac{1}{nm}, \frac{1}{nm^2}, ...\}$$
$$\mathcal{Z}_{n,m} = \{\mathcal{H}_{n,m}, \mathcal{G}_{n,m}\}$$

Li and Cheng [1] have shown that the schedulability bound using the MulZ algorithm with a First Fit Decreasing resource allocation strategy will never exceed 0.5. Li and Cheng [2] also found using Adjusted Availability Factor (AAF) and $n = \{2, 3, 4, 5, 7\}$ that these numbers can be swapped into the same level, significantly reducing the schedulability bound. When using this set of numbers for $n$ in the MulZ algorithm, we get $\mathcal{Z}_{4,2} = \mathcal{Z}_{2,2}$, narrowing the Z-approximations down to $\mathcal{Z}_{n,2}$ with $n = \{3, 4, 5, 7\}$. The idea of using MulZ-FFD is to approximate with all of these sequences simultaneously and choose which one to use to assign partitions to the resource-units. The pseudocode for MulZ [1] is reproduced here:

```
(0) res_units R := {R_j{factor = 0, rest = 1} : j ϵ [1, m]};
(1) partitions P := {P_j{weight, res_unit = 0} : j ϵ [1, s]};

(2) bool MulZ_FFD ()
(3)   sort P in non−increasing order;
(4)   for j = 1 to s do
(5)     P_j.resource := MulZ_FFD_Alloc(P_j.weight);
(6)     if P_j.resource = 0 return false;
(7)   od
(8)   return true;

(9) int MulZ_FFD_Alloc (w)
(10)  for i = 0; n ϵ {3, 4, 5, 7}; i++ do
(11)    A_i = R_{Z_{n,2}}(w);
(12)  do
(13)  for k = 1 to 4 do
(14)    r := the k−th minimum item in array A;
(15)    f := n, where w is approximated at r by Z_{n,2};
(16)    for j = 1 to m do
(17)      if R_j.factor = f and R_j.rest ≥ r do
(18)        R_j.rest := R_j.rest − r;
(19)        return j;
(20)      od
(21)      else if R_j.factor = 0 do
(22)        R_j.factor := f;
(23)        R_j.rest := 1 − r;
(24)        return k;
(25)      od
(26)    od
(27)  od
(28)  return 0;
```

We now discuss our concerns with RT-Xen and our ideas on extending support for them. This will be a large and challenging project on its own, due to the fact that partitions in RT-Xen as it currently stands are defined as the VCPUs assigned to PCPUs. For the RRP model, partitions are defined by time-slices, and when partitioning a PCPU each partition will have a different computing power. This is beneficial as if we had one VCPU to a PCPU, we are effectively partitioning that VCPU into multiple partitions. This allows for multiple tasks to run simultaneously rather than what would have been running sequentially, but with less computing power. Our partitions may not be sliced in a neat way and may even be co-prime, so we cannot support this just by finding proportions of VCPUs to assign to the same PCPU.

The next problem to address is the one of VCPU migration across PCPUs. As briefly mentioned earlier, RT-Xen binds multiple VCPUs to individual PCPUs and runs tasks on these. Unfortunately, there is no way to allow the migration of VCPUs between PCPUs in the current state without masking, let alone our partitions. Developing a form of support for this onto RT-Xen will allow for even basic migration between CPUs. An initial proposition for this VCPU migration is to ensure no tasks are currently being ran on the VCPU and/or partition, then effectively changing its pin from one PCPU to another.

Alas, one easily overlooked concern which may cause problems in the distant future is the compatibility with computer architectures. Hardware is getting exponentially more complex and powerful as time proceeds, making it hard for development on these platforms to keep up especially if not actively maintained. Surely it is possible to obtain earlier versions of Linux and Ubuntu when necessary, but once the current systems are inevitably replaced we must ensure compatibility in the case of having to use older software versions. This is not ideal, but the quick solution is to use older hardware with the older software in case of neglected updates moving forward. Once MulZ is implemented, the platform is there forever so the only thing to concern ourselves with is updates to the architecture so that our algorithms will push on with the technology, perhaps yielding even better results. Our potential solutions to these are outlined in the next section.

## III. Implementation Strategies

First, the most important challenge for the MulZ algorithm's implementation on RT-Xen is the lack of a partition definition on the platform. We are planning to implement partitions to these PCPUs based on time-slices using a table-driven scheduler. This scheduler will consist of a data structure with three members, and will make use of interrupts during the execution of tasks in order to allocate time to these partitions accurately. This is of high priority as it is not only required to implement MulZ, but even the single-CPU Magic-7 partition scheduling algorithm from Li and Cheng [2]. Once this issue is resolved we will implement Magic-7, which partitions a single CPU into partitions the exact way we have described: based on time-slices given weights through these approximation sequences and MulZ.

To obtain the approximations to these partitions via MulZ, we should start by looking at a set of partitions sorted in decreasing order: $G = \{0.65, 0.60, 0.55, 0.50, 0.35\}$. Using $\mathcal{Z}_{n,2}$, we can pass in the first partition 0.65 and use $n = 3, 4, 5, 7$ to get the approximated weights as $\frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{7}$ respectively. Once these are calculated we take the minimum value, $\frac{2}{3}$, and use the $n$ of its E-ABS for the first PCPU. Doing

this $\forall w \ \epsilon \ G$ until the last element, we see the remaining PCPUs become occupied and have different working E-ABSes. Repeating this process with 0.35, the approximated weights are $\frac{2}{3}, \frac{1}{2}, \frac{2}{5}, \frac{3}{7}$, and our working E-ABS is $\mathcal{Z}_{5,2}$, assigning this partition to PCPU-2. An extended implementation is shown in Figure 1, where we use the MulZ algorithm to assign these partitions appropriately. With this strategy, we can get more partitions from a PCPU than we would have by assigning VCPUs, allowing more tasks to run simultaneously.

Upon completion of this preparation, we will need to build and compile a new kernel for the dom0 with RT-Xen to run the Magic-7 scheduler. Once this step is completed, the goal is to begin implementing the MulZ algorithm and a similar kernel will need to be created, except to do this the bigger issue of task migration across PCPUs must also be resolved. We must dive into how RT-Xen is managing its scheduling for these PCPUs and try to translate the allotment of these VCPUs into VCPUs that are quantum-based. A tracker for the program counter will likely need to be included and obtaining the hyperperiod of the taskset may be required, thus something along the lines of $counter\%hyper\_period$ will prove useful to guarantee accurate scheduling and interrupting.

Another issue to consider is that of overhead. While the algorithms discussed do take this into consideration, our development of an implementation must consider the overhead of not only the algorithms, but the interrupts and migrations as well. Perhaps when migrating multiple instances of tasks on different PCPUs to the same one, or interrupting the scheduling of a task once the quantum reaches our allotted amount, overhead is there. Because of this, we have investigated using a hash-table for our table-driven scheduler to ensure an efficient time complexity.

The final issue we have to consider is not one involving our implementation, but one of compatibility. With the hardware evolving at increasing rates, and the last update to RT-Xen being in 2015 based on Xen 4.6 which is much further along now, we must consider the operating systems. A common setup for RT-Xen usually consists of Ubuntu 12 or 14, and 16 has been known to cause problems during the installation. While this may not occur until the distant future, being bottle-necked by compatibility issues with operating system distributions is far from ideal. Since RT-Xen is open-source, we will try to help with this issue going forward, but our priority is set on this implementation with the current setups.

Once these major issues are resolved, we will begin our testing with LITMUS$^{RT}$ running on the guest domains and our own previously mentioned kernel on the host. This provides us a way to generate tasks on the guests and allow our host to manage how these tasks are scheduled through our own algorithms, with MulZ being the final goal in this initial step towards a regularity-based model in the real world.

## IV. Conclusion

Up to this point we have discussed our MulZ algorithm with its associated pre-requisite knowledge and issues of implementation. Even though the optimality of this MulZ algorithm has not been proven theoretically, the experiments conducted have shown to be very positive [1]. Schedulability and resource utilization using MulZ-FFD have displayed drastic improvements over the more commonly used periodic counterparts.

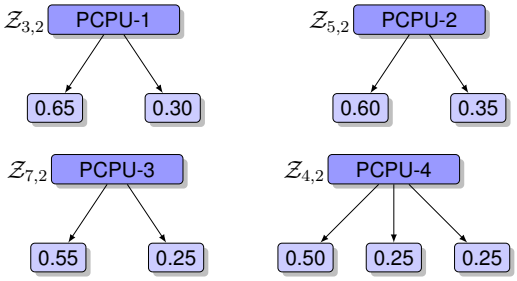



Fig. 1. Planned RT-Xen Partition Implementation

Our proposed plan of implementation is to first define partitions by time-slices on RT-Xen rather than entire VCPUs. We will then follow this with this creation of a table-driven scheduler to help manage the task migration across CPUs, while also being able to identify and resolve multiple instances of a task running on different PCPUs. Upon completion of these steps, we will minimize the overhead of this migration and any potential delays from our implemented table-driven scheduler such as interrupts. Finally, we will work towards future-proofing our scheduler as well as RT-Xen by resolving the issues of compatibility with later Linux and Ubuntu versions.

The prevalence of real-time systems is becoming much more known as we gear up for future technologies such as autonomous cars. Thus, the importance of a real-world application for better use of resources given to guarantee better schedulability and utilization cannot be understated. HiRTS allow for the sharing of a resource's computational power, and MulZ-FFD is aiming to enhance this capability on RT-Xen. We look forward to presenting concrete results at a later date, making significant advances towards a better model in the real-world.

## References

[1] Y. Li and A.M.K. Cheng, "Toward a Practical Regularity-based Model: The Impact of Evenly Distributed Temporal Resource Partitions" (ACM TECS '17)
[2] Y. Li and A.M.K. Cheng, "Static Approximation Algorithms for Regularity-based Resource Partitioning" (IEEE RTSS '12)
[3] S. Xi, M. Xu, C. Lu, L.T.X. Phan, C. Gill, O. Sokolsky, I. Lee, "Real-Time Multi-Core Virtual Machine Scheduling in Xen" (ESWEEK '14)
[4] S. Xi, J. Wilson, C. Lu, C. Gill, "RT-Xen: Towards Real-time Hypervisor Scheduling in Xen" (EMSOFT '11)
[5] A.K. Mok, X. Feng, D. Chen, "Resource Partition for Real-time Systems" (RTAS '01)
[6] LITMUS-RT (http://www.litmus-rt.org)
[7] Xen Project (https://www.xenproject.org)

# Towards a model-based framework for prototyping performance analysis tests

Thanh-Dat NGUYEN
LIAS-ENSMA
Futuroscope, France
Email: thanh-dat.nguyen@ensma.fr

Yassine OUHAMMOU
LIAS-ENSMA
Futuroscope, France
Email: yassine.ouhammou@ensma.fr

Emmanuel GROLLEAU
LIAS-ENSMA
Futuroscope, France
Email: grolleau@ensma.fr

***Abstract*—In this paper we propose a framework dedicated to integrate analysis tests into a model-based development chain of critical RTES (Real-Time Embedded Systems) by easing their prototyping to conclude about the temporal performance. The framework allows to generate executable programs in a target specific language (e.g., Scilab) of real-time scheduling tests from formal representations of their algebraic mathematical formulas with precise semantics. Such a framework is beneficial because of the continuous evolution of RTES architectures, programming methodologies and analysis techniques. Our framework is composed of three components: *mathematical formulas representation*, *real-time semantics mapping* and *code generation*. The framework relies on standard and common languages: AADL (Architecture Analysis Design Language) for real-time semantics, MathML for formulas representation and Scilab for target executive language. The development of this framework is based on model-driven engineering settings.**

## I. Context and Problem statement

Nowadays, the scheduling analysis of RTES has been set up in the model-based development life-cycle as an independent phase that follows immediately and interacts with the modelling phase. On the one hand, that allows RTES designers to conclude about the performance of their designs as early as possible to avoid errors that can impact sharply the development costs. On the other hand, that facilitates the use of the analysis tests, enhances the usability of real-time scheduling and bridges the gap between the theory and practical cases. Several scheduling tests have been proposed based on algebraic mathematical formulas, such as but not limited to response-time analysis (RTA) [1], [2], demand bound function (DBF) [3], request bound function (RBF) [4], etc. There are several tools that implemented some real-time analysis tests (e.g., Cheddar [5], MAST [6], Rt-Druid [7], etc.). Thanks to model-driven engineering settings the utilization of these tools facilitates more and more performing scheduling tests through the model-based development, but that can be enhanced due to the problems discussed hereafter.


Fig. 1. Modeling, transformation and analysis via a model-based work-flow

Unfortunately, each analysis tool has its own approach and presents a specific temporal analysis viewpoint and specific real-time workload model. Then, to validate a system, we generally have to transform the system design (expressed in AADL[8] or MARTE[9]) into the formalism of a chosen target tool (as it is shown in Figure 1). (i) This is a costly step because it requires a deep understanding of each tool approach. Also, when we transform, we may lose in term of precision since we adapt models to analysis tools (because of the semantics gap between the modelling languages and the analysis tools [10]–[12]). Moreover, we generally need a reverse transformation to enrich the design by the obtained analysis results. (ii) Moreover, new proposed tests are often developed as homemade prototypes, which are not immediately and evidently transferable to the industry or integrated in the model-based tool chain. (iii) Also, the analysis tools are not always extensible easily since they are hard-coded, and even for extensible tools the extension (e.g., adding new analysis tests based on specific analysis viewpoint) may lead to get other semantics for the same parameter, hence it can lead to a redundant or conflictual meaning at long-term.

In this paper, we aim to bring analysis tests to the modeling area instead of transforming design models to analysis tools area. To do so, we propose a model-based framework enabling RTES analysts and researchers to prototype and integrate their analysis tests in order to be shared and use easily by designers. This framework can also be used as a teaching-aid tool. In the following: Section II presents our proposition with a proof of concept and Section III concludes the paper.

## II. Our contribution

### A. Approach

This part presents a simple motivation example to highlight requirements that the proposed framework should meet.

*Motivation and running example:* The following mathematical formula represents the response time analysis test as proposed in [1] and which we will use as a running example throughout the rest of the paper (particularly in the proof of concept section).

$$R_{i.m} = \begin{cases} C_i & m = 0 \\ C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_{i.m-1}}{T_j} \right\rceil C_j & otherwise \end{cases} \quad (1)$$

$$hp(i) = \{k | 1 \leq k \leq n \wedge Pr_k > Pr_i\} \quad (2)$$

The worst case response-time of each task $i$ is computed according to Eq. 1, where C, T, and hp represent respectively the worst case execution-time, the period and the set of tasks their priorities are higher than the priority of the task $i$.

We assume that an analyst or researcher would like to share the test (of the running example) in order to be used by designers (non-expert in real-time scheduling analysis) during the modeling phase. To do so, we need a collaborative framework ($\Gamma$) (dedicated to analysts and designers according) to the following requirements.

*Requirement 1:* The framework $\Gamma$ shall offer a descriptive mathematical formalism independent from the real-time semantics and machine-interpretable. The formalism must be sufficiently expressive and concrete to exactly express all typical real-time mathematics equations.

*Requirement 2:* The framework $\Gamma$ shall provide support to define semantic mapping relationships between mathematical formulas and different RTES design languages. Since the mathematical formulas are intended to be independent from the real-time semantics, to have input data for the computation, we need to map each element (quantitative parameter) of the mathematical formula to an element type (class) from the design language (e.g, MARTE or AADL). Hence, that enables to specify the real-time semantics of the parameter referring to the design language used for modelling.

*Requirement 3:* The framework $\Gamma$ shall provide an engine relying on *Requirement 1* and *Requirement 2* enabling to generate the executable code in order to perform the analysis tests for each system design expressed in supported design languages.

### B. Architecture of the proposed framework

Figure 2 depicts the overview of the proposed framework, which is based on three main components: *math formulas representation*, *real-time semantics mapper* and the *code generation*.

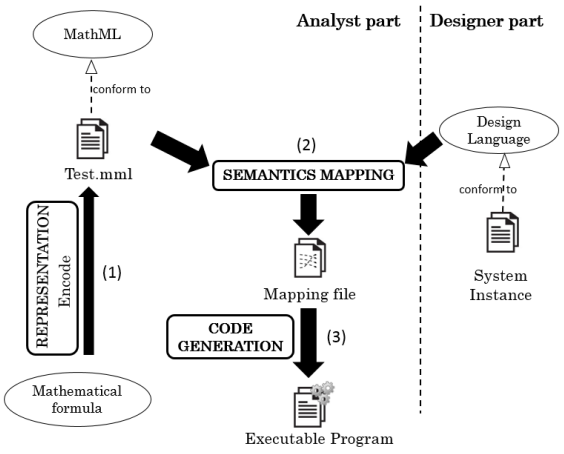



Fig. 2. Framework architecture

*1) Formula Representation Component:* We have opted for MathML (*Mathematical Markup Language* [13]) since it is a standard for describing mathematical notations and capturing both their structures and contents. This component allows (action (1) of Figure 2) to researchers/analysts to express their analysis tests conforming to MathML meta-model as it is sketched in Figure 3. Thanks to this component, we can obtain a repository of tests where each test is represented by MathML file.

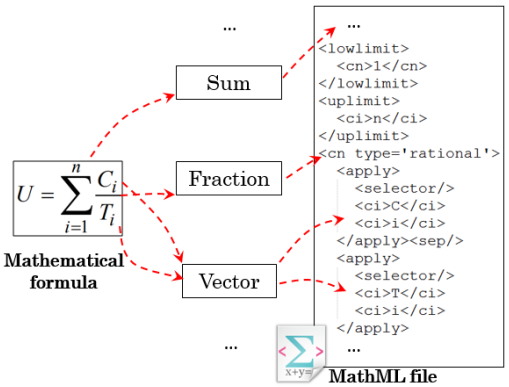



Fig. 3. Formula representation component

*2) Real-time Semantics Mapping Component:* Every parameter of a given test expressed as MathML file can be mapped to a real-time concept existing in a given design language. Therefore, the *semantics mapper component* is dedicated to analysts to define the mapping relationships between their tests and design languages (action (2) of Figure 2). Figure 4 shows the meta-model permitting to map an analysis test to different design languages. The idea behind this "mapper" is to make the analysis tests available for users who are familiar with different design languages. Then, each instance conforms to the mapper meta-model relies on a specific test and a specific design language.

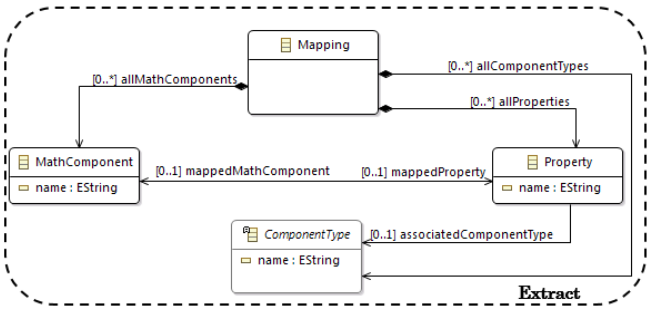



Fig. 4. Excerpt of Meta-model expressed in Ecore[14] of semantics mapping component

*3) Execution code generator:* Once the test is expressed as MathML file, the mapping relation with a specific design language is defined the generator can be developed by analysts based on a template and according to a specific target language. This template will be used by the designer once the system (under-analysis) is modelled with the same specific design language, then the code is not hard-coded and can

be generated according to the target programming language (action (3) of Figure 2). The execution of generator component allows to generate the code during the modeling phase at run-time conforming to several programming languages (e.g. Python, Matlab, etc.).

### C. Proof of Concept

In this part we show the usability of our framework through a generic scenario usage. We choose (***Architecture Analysis and Design Language***)[1] as a design language and Scilab as a programming language. We assume that an analyst/researcher aims to share with the community and with RTES designers the test presented in the motivation example (see sub-section A of Section 2). First, the test need to expressed in MathML. Secondly, the analyst should define a mapping relation between the test concepts and AADL. Figure 5 shown the test parameters and their associated concepts.

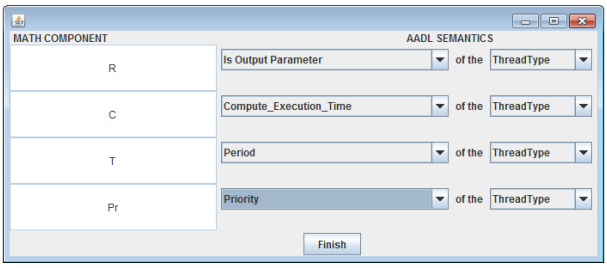



Fig. 5. Mapping process

We have developed a code generator according to Scilab syntax. We also assume that a designer, using AADL to conceive a system, chooses the MathML file and launch the execution. The developed code generator will transform the test and numeric values of the designed system to a Scilab program as it is shown in Figure 6.

## III. Conclusion

This papers introduces a work-in progress. It sketched how to prototype easily and rapidly new tests to be integrated in model-based tool chain, whilst avoiding difficulties related to technological backgrounds and problems of analysis and modeling environments compatibility. The resulting framework will be shared with the community under LGPL licence, and it will be enriched to support other programming languages. This work will be completed by being associated to the work presented in [15] to help designers by orienting them to choose the tests that match their needs. We believe that this kind of work will help the community to capitalize tests for reuse and also learning purposes. In this perspective, real-time conferences set up recently an artifact evaluation dedicated to "accepted-for-publication" papers. This evaluation consists of reproducing the results of the accepted papers by a conference committee by using mediums like virtual machines or codes accompanied by textual guidelines. However, the current reproducibility process lacks to store the test itself and data

[1] www.openaadl.org

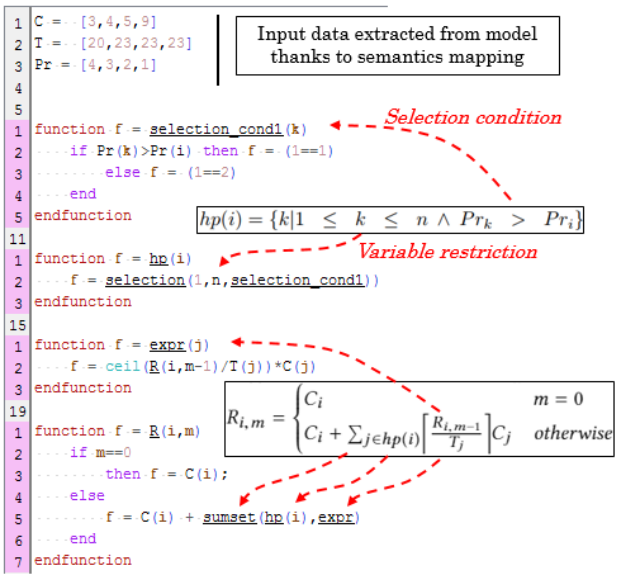



Fig. 6. Response Time Analysis in Scilab

related to when and how to use the test in order to conclude about the temporal behavior of critical systems applied to this test.

## References


[1] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[2] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Real-Time Systems Symposium, 1990. Proceedings., 11th*. IEEE, 1990, pp. 201–209.

[3] K. Jeffay and D. Stone, "Accounting for interrupt handling costs in dynamic priority task systems," in *Real-Time Systems Symposium, 1993., Proceedings.* IEEE, 1993, pp. 212–221.

[4] N. Fisher and S. Baruah, "A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines," in *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*. IEEE, 2005, pp. 117–126.

[5] "The cheddar project : a gpl real-time scheduling analyzer," http://beru.univ-brest.fr/ singhoff/cheddar/, 2015, [Online; accessed 15/05/2018].

[6] MAST, "Modeling and analysis suite for real-time applications," http://mast.unican.es/, [Online; accessed 15/05/2018].

[7] RT-Druid, http://www.evidence.eu.com/products/rt-druid.html.

[8] AADL, "Architecture analysis and design language," http://www.aadl.info/aadl/currentsite/.

[9] MARTE, "Modeling and analysis of real-time and embedded systems," http://www.omg.org/omgmarte/, [Online; accessed 15/05/2018].

[10] C. Mraidha, S. Tucci Piergiovanni, and S. Gerard, "Optimum: a marte-based methodology for schedulability analysis at early design stages," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 1, pp. 1–8, 2011.

[11] Y. Ouhammou, E. Grolleau, P. Richard, and M. Richard, "Reducing the gap between design and scheduling," in *RTNS*, 2012, pp. 21–30.

[12] R. Henia, L. Rioux, and N. Sordon, "Demo abstract: TEMPO: integrating scheduling analysis in the industrial design practices," in *2016 IEEE RTAS*, 2016, p. 63.

[13] Wikipedia, "Mathml," https://en.wikipedia.org/wiki/MathML, [Online; accessed 15/05/2018].

[14] Ecore, "Eclipsepedia," https://wiki.eclipse.org/Ecore.

[15] T. D. Nguyen, Y. Ouhammou, and E. Grolleau, "Parad repository: On the capitalization of the performance analysis process for aadl designs," in *European Conference on Software Architecture*. Springer, 2017, pp. 22–39.