RTSOPS 2011

Proceedings of the 2nd International Real-Time Scheduling Open Problems Seminar

Porto, Portugal
July 5, 2011

In conjunction with:
The 23rd Euromicro Conference on Real-Time Systems (ECRTS11),
July 6-8, 2011

Edited by Robert I. Davis and Nathan Fisher

Foreword

Welcome to Porto and the 2nd International Real-Time Scheduling Open Problems Seminar (RTSOPS 2011). This seminar provides a venue for the exchange of ideas and the discussion of interesting unsolved problems in real-time scheduling. The format of the seminar positively encourages interaction between participants and provides ample time for relaxed discussions. The goal of the seminar is to promote a spirit of co-operation and collaboration within the real-time scheduling community.

RTSOPS 2011 is organized around presentation and collaboration sessions. Each presentation session provides the opportunity to hear about a number of important unsolved problems in real-time scheduling, highlighted via brief presentations. The following collaboration session gives participants the opportunity to interact in small groups, exchanging ideas with the presenters about how the problems might be solved, and to take the first steps towards a solution.

A total of 12 open problems were selected for presentation at the seminar, along with a status report on an open problem from the previous year. These proceedings are also published as a Technical Report from the University of York, Department of Computer Science (YCS-2011-466) available at http://www.cs.york.ac.uk/ftpdir/reports/2011/YCS/466/YCS-2011-466.pdf.

We would like to thank the Steering Committee listed below, for their work in reviewing the open problems, and helping to make the seminar a success.

Björn Andersson	Software Engineering Institute, Carnegie Mellon University (USA)
Marko Bertogna	Scuola Superiore Sant'Anna, Pisa (Italy)
Liliana Cucu-Grosjean	INRIA Nancy-Grand Est (France)
Shelby Funk	University of Georgia (USA)

Special thanks also go to Karl-Erik Arzen, Eduardo Tovar, Stefan Petters, and Gerhard Fohler for their support and assistance in organising this seminar.

Robert Davis and Nathan Fisher
Co-chairs
2nd International Real-Time Scheduling Open Problems Seminar (RTSOPS 2011)

Table of Contents

Influence of the Task Model on the Precision of Scheduling Analysis for Preemptive Systems

Status Report

Sebastian Altmeyer, Saarland University, altmeyer@cs.uni-saarland.de
Claire Maiza, INP Grenoble, VERIMAG, Claire.Maiza@imag.fr

Problem Statement

In last year's RTSOPS workshop, we raised several questions along the interface between timing analysis and scheduling analysis for preemptive systems.

The interface between these two analyses is the task model constituting an abstraction of the task's timing properties in the system. A simple task-model exhibits a single value for the execution demand. In preemptive systems, this value then must contain the preemption cost overhead—independent of the number of preemptions and any additional information. Schneider [7] proposed such a timing analysis including preemption cost. Timing analysis, however, can compute, in addition to the pure time bound for uninterrupted execution, the additional delay due to interrupts or preemptions. These costs strongly depend on the specific preemption points and on the preempting task; preemption costs may vary from nearly zero to large fractions of the task's execution time. Thus, timing analysis may compute not only an upper bound on the preemption costs for a task i but also additional bounds for preemption of task i by task j [2], [9], or for the n^{th} preemption of task i, or for preemption occurring at point p [1], [6]. The tradeoff between precision and complexity of the schedulability analysis is determined by the task model and its abstraction of the timing of tasks.

Open question in this area are:

- What is a useful abstraction level?
- Which task model to use; what is a good tradeoff between precision of the analysis and complexity?
- How to integrate detailed information of preemption cost within the schedulability analysis?

Towards solutions to these open problems, we have in a first step worked on a precise view of task model and the cooperation of timing and scheduling analysis. We have also classified the existing work that includes preemption costs according to the applied task model. While in case of fixed priority scheduling, several approaches [10], [8], [3], [6] exist that include the preemption cost explicitly, most of them using different task models, there is only one [5] for dynamic priorities published this far.

Interaction Timing Analysis/Scheduling Analysis

We denote the collection of all input variables to the schedulability analysis as a *task model*. A task model includes at least execution time bounds, periods, deadlines and priorities. Within the complete timing verification (including timing analysis and schedulability analysis), different task models may be used. Figure 1 shows the basic verification process: the task model as the interface between timing analysis/system constraints and scheduling analysis.

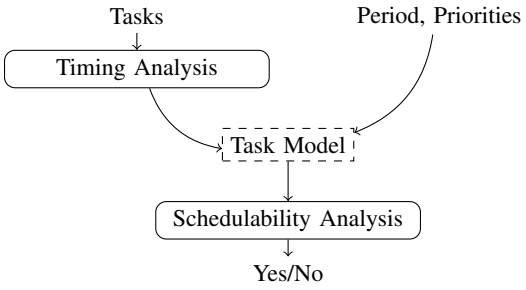


Fig. 1. Verification process with basic task model only

In some cases, the task model delivered by the timing analysis and the one used by the scheduling analysis may differ; a precomputation step may then transform the first task model to the derived one (see Figure 2)—this precomputation may iteratively refine the task model based on the preliminary schedulability results (see Figure 3). In view of these verification processes, we need to define two different task-models: *basic task model* and *derived task model*.

Definition (Basic Task Model) *Basic task model consists of (a) information by the overall system (period, priority) and (b) the set of all values directly delivered by the timing analysis and used in subsequent steps.*

Definition (Derived Task Model) *Derived task model consists of all direct input-values to the schedulability analysis.*

For instance, assume task i with execution demand C_i can be preempted at most b times and the preemption cost for such a preemption is given by p_i. The basic task model with implicit preemption cost is then given by $C'_i = C_i + b \cdot p_i$. An example of the iterative refinement (as in Figure 3) is given, if bound b is adapted depending on the results of the schedulability analysis, i.e. depending on preciser response time estimations. Note that the precomputation may be iterative, i.e., may be refined during the schedulability analysis. A complete separation of timing analysis and scheduling analysis is not possible. Precise bounds for a specific (possibly nested) preemption scenarios

may be needed by the scheduling analysis—precomputation of all possibly needed scenarios is computationally infeasible.

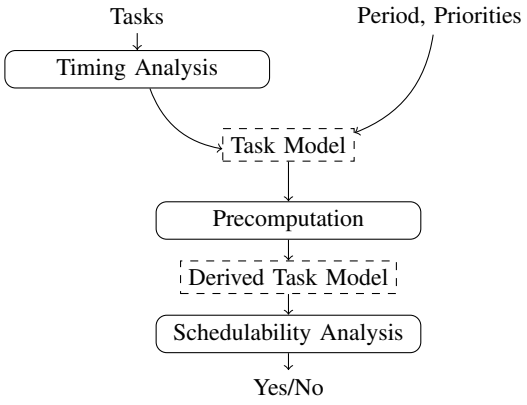


Fig. 2. Verification process with basic and derived task model

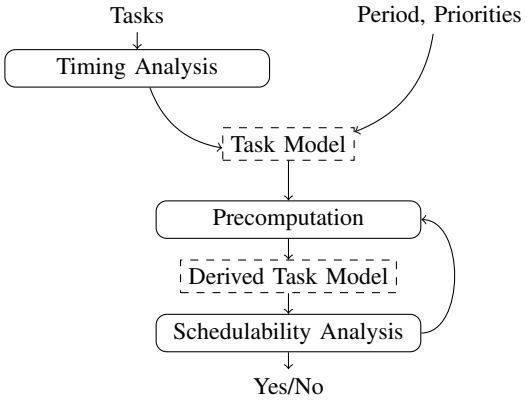


Fig. 3. Verification process with iteratively refined derived task model

Note that the critical instance, i.e., the activation scenario causing the highest tasks' response times depend on the chosen task model. Synchronous release times—the worst-case scenario in Liu and Layland's simple task model—does not necessary lead to the highest preemption cost. Releasing all tasks consecutively with short offsets and in decreasing priority order usually leads to higher preemption costs. For precise preemption costs taking the effect of preempting and preempted task into account, it can be proven that even this scenario is not the critical instance.

Fixed Priority Scheduling

In case of fixed priority scheduling, the common response time analysis [4] has been extended to include preemption cost explicitly. We classified these different analyses and identified the basic and derived task-model (as depicted in the figures). Note that we omit this list due to space limitations. The analyses often only differ in the used task model. Although we cannot determine the inherent imprecision of the taskmodel by this, we can evaluate the precision of the different analyses. As a first result, we have seen that information about the nth preemption of a task provide little or no additional precision. The reason is that preemption points exhibiting the worst-case preemption cost often occur within loops. Thus, this approach may lead to preciser results only when the number of preemptions is greater than the number of iterations of such a loop. Furthermore, incorporation of such information requires a high degree of computational overhead and thus, does not pay off. We plan to conduct extensive comparisons of the different analyses based on random test-cases and case-studies.

REFERENCES

[1] S. Altmeyer and C. Burguière. A new notion of useful cache block to improve the bounds of cache-related preemption delay. In *ECRTS 2009*.

[2] S. Altmeyer, C. Burguière, and J. Reineke. Resilience analysis: Refinement of the CRPD bound for set associative caches. In *LCTES 2010*.

[3] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *RTAS'96*.

[4] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, May 1986.

[5] L. Ju, S. Chakraborty, and A. Roychoudhury. Accounting for cache-related preemption delay in dynamic priority schedulability analysis. In *DATE 2007*.

[6] C.-G. Lee, J. Hahn, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. In *RTSS 1996*.

[7] J. Schneider. *Combined Schedulability and WCET Analysis for Real-Time Operating Systems*. PhD thesis, Universität des Saarlandes, 2003.

[8] J. Staschulat and R. Ernst. Scalable precision cache analysis for real-time software. *ACM TECS*, 6(4):25, 2007.

[9] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *ECRTS 2005*.

[10] Y. Tan and V. J. Mooney. Timing analysis for preemptive multi-tasking real-time systems with caches. In *DATE'04*.

How Many Boundaries Are Required to Ensure Optimality in Multiprocessor Scheduling?

Geoffrey Nelissen[†1], Shelby Funk[‡], Dakai Zhu[§], Joël Goossens[†]

[†]PARTS Research Center
Université Libre de Bruxelles (ULB)
Brussels, Belgium

[‡]Department of Computer Science
University of Georgia
Athens, GA, USA

[§]Department of Computer Science
University of Texas
San Antonio, TX, USA

I. Introduction

To the best of our knowledge, all current optimal scheduling algorithms for multiprocessor platforms divide the time into intervals. Within each interval, a predefined set of scheduling rules is applied to schedule the task set. At each interval *boundary*, the tasks' characteristics are updated so that, according to the set of scheduling rules, all tasks can meet their deadlines without parallelism during the next time interval. These boundaries are needed to ensure the correctness of the schedule. Unfortunately, they introduce many time overheads by increasing the number of scheduling points, preemptions and migrations.

All recent optimal scheduling algorithms impose such boundaries at every job's arrival and/or deadline (e.g., [1]–[4]). We believe that the number of required boundaries is currently overestimated and we wonder how many of them are really necessary to ensure the optimality.

II. Model

We consider a set of n periodic tasks with implicit deadlines scheduled on m identical processors. Each task τ_i in the set τ is characterized by its worst case execution time C_i and its period T_i ($C_i \leq T_i$). That is, τ_i releases an infinite number of jobs separated by T_i time units and each job must execute during C_i time units before the next job arrival. We define the utilization of τ_i as $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$. The total utilization of the system is the sum of all the tasks's utilization; i.e., $U \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i$. We say a task set τ is feasible on m processors if $U \leq m$ and $U_i \leq 1$ for all $\tau_i \in \tau$.

Note that, according to the boundary-based scheduling rules, tasks may be preempted and migrate between processors.

III. Problem Statement

There are multiple optimal multiprocessor scheduling algorithms for periodic tasks [1]–[5] (i.e., algorithms capable to schedule any task set τ with $U \leq m$ and $U_i \leq 1 \ \forall \tau_i \in \tau$ without deadline miss or intra-job parallelism). All optimal multiprocessor scheduling algorithms are executed using time slices. These time slices are bounded by two consecutive *boundaries*. As such, these schedulers have two phases —a boundary phase and an execution phase. During the boundary phase, local execution parameters are allocated to each task and/or each processor. During the execution phase, tasks are executed and each task and/or processor must complete its allocated local execution before the next boundary.

With the exception of the EKG algorithm proposed by Andersson and Tovar in [3], all recent optimal multiprocessor scheduling algorithms impose boundaries at the jobs' deadlines, even if the number of processors m is equal to 1. However, the *earliest deadline first* (EDF) algorithm correctly schedules any task set with $U \leq 1$ on a uniprocessor platform, without modifying the tasks' properties during the schedule [6]. Hence, boundaries are obviously not required when $m = 1$. We could thereby reasonably wonder if they are really needed when m is greater than 1. We strongly believe that the answer to this question is "yes". Indeed, a new constraint arises when the number of processors is greater than 1: a job cannot be executed on two (or more) processors simultaneously. Hence, time boundaries play the role of synchronization points between tasks, guarantying that no intra-job parallelism will be required during the next time interval to respect all jobs' deadlines. However, many scheduling algorithms have been proposed for multiprocessor platforms without using the boundary approach [7], [8]. Even though they are no longer optimal, they successfully schedule most task sets.

Therefore, the open question becomes: how many time boundaries should be added to ensure the optimality?

IV. A First Conjecture

Consider a system where tasks are categorized as *migratory* and *non-migratory*. A non-migratory task is statically assigned to a processor π_j and can only be executed on this processor. On the other hand, a migratory task can migrate between multiple processors. If we assume that there is at most one non-migratory task per processor and $m-1$ migratory tasks in the system (i.e., there are at most $(2 \cdot m) - 1$ different tasks in τ), then we have the following conjecture:

Conjecture 1: Assume τ is a task set that is feasible on m processors and contains at most $2 \cdot m - 1$ tasks. If one task is statically allocated to each processor and the remaining tasks are permitted to migrate, then there exists an optimal algorithm

[1]Supported by the Belgian National Science Foundation (F.N.R.S.) under a F.R.I.A. grant.

Tasks					
Non-migratory			Migratory		
	C	T		C	T
N_1	2	6	M_1	10	12
N_2	4	8	M_2	16	24
N_3	16	24			

TABLE I
CHARACTERICTS OF TASKS USED IN FIG. 1.

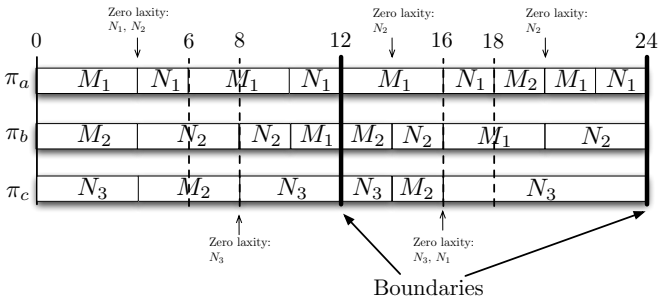


Fig. 1. Schedule using Rule Set 1, when only considering deadlines of migratory tasks as boundaries.

that can schedule τ using boundaries only at the migratory task deadlines.

To illustrate Conjecture 1, we propose an example. Consider three non-migratory tasks N_1, N_2 and N_3 and two migratory tasks M_1 and M_2 with the characteristics given in Table I. This task set τ has six distinct job deadlines within its hyper-period (i.e. the least common multiple of all task periods), but only two distinct migratory task deadlines (at time 12 and 24). We show in Fig. 1 that we can successfully schedule τ with Rule Set 1 when considering only the migratory task deadlines as boundaries. Note that we do not say that Rule Set 1 will succeed in the schedule of all task sets. However, we believe that it is a good starting point for future investigations.

***Rule Set** 1:*

1) Every task τ_i must execute $U_i \times L$ time units between two consecutive boundaries separated by L time units (i.e. between two migratory task deadlines).
2) A migratory tasks M_i has always a higher priority than a non-migratory task N_k, unless N_k has a zero laxity.
3) Any job with a zero laxity must be executed immediately.

The main idea behind Rule Set 1 is the following:
By definition, there is no risk that a non-migratory task could be executed simultaneously on two different processors. Therefore, we must execute the migratory tasks whenever it is possible to schedule them without parallelism. Hence, a migratory task M_i must have a higher priority than a non-migratory task N_k, unless N_k will miss its deadline if it is not executed immediately.

V. OPEN PROBLEMS

- Assuming that Conjecture 1 is true, could we directly extend this result to the general case where there are an arbitrary number of non-migratory tasks on each processor?
- Could we still reduce the number of time boundaries or does the number of migratory tasks' deadlines constitutes a lower bound?
- Could we extend this approach for sporadic tasks?

These are a few questions for which we do not have answers yet. However, these results could have a strong impact on the design of future multiprocessor schedulers. Indeed, it could help to reduce the run-time overheads during the schedule and hence improve the practicality of future multiprocessor scheduling algorithms.

REFERENCES

[1] D. Zhu, X. Qi, D. Mossé, and R. Melhem, "An optimal boundary-fair scheduling algorithm for multiprocessor real-time systems," *Journal of Parallel and Distributed Computing (accepted, to appear)*, 2011.
[2] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-Fair: A simple model for understanding optimal multiprocessor scheduling," in *ECRTS '10*, July 2010, pp. 3–13.
[3] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," *RTCSA '06*, pp. 322–334, 2006.
[4] S. Funk and V. Nadadur, "LRE-TL: An optimal multiprocessor algorithm for sporadic task sets," in *RTNS' 09*, October 2009, pp. 159–168.
[5] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.
[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[7] S. Kato and N. Yamasaki, "Portioned EDF-based scheduling on multiprocessors," in *EMSOFT '08*, 2008, pp. 139–148.
[8] A. Burns, R. Davis, P. Wang, and F. Zhang, "Partitioned EDF scheduling for multiprocessors using a C=D scheme," in *RTNS '10*, 2010, pp. 169–178.

How Different are Offline and Online Scheduling?

Gerhard Fohler, University of Kaiserslautern, Germany

The heated debate on whether offline or online scheduling (e.g., [7], [2], and [9]) is preferable for real-time systems has taken attention away from the question how the two are actually defined and what their essential differences are. Results and arguments in the debate are typically based on a conglomerate of assumptions, system design issues, and particular views of scheduling rather than the actual cores of the ways to schedule themselves. We do not wish to get involved into the "TT' vs. ET" debate, rather have a look the related scheduling paradigms to determine what their differences are. We will argue they are much smaller than perpertuated clichees.

One of the central choicesfor scheduler design is that of the *activation paradigm*, i.e., when are events recognized, who initiates activities, when are these decisions taken? In conventional systems, the *event triggered* approach is prevalent, in which occurrences of events initiate activities in the system immediately. In *time triggered* systems, activities are iniated at predefined points in time [7].

I. Offline Scheduling

Initiating activities in the system with the progression of time requires complete understanding of the system and the environment it will operate in. Scheduling for TT is usually carried out via a scheduling table, with tasks and activation times. An offline algorithm takes complete information about the system activities, which reflect the knowledge about anticipated environmental situations and requirements, and creates a single table, representing a feasible solution to the given requirements. As the algorithm is performed offline, fairly complex task sets can be handled, e.g., precedence constraints, distribution and communication over networks, task allocation, mutual exclusion, separation of tasks, etc. - e.g. [8], [3], [5], [6], [4], [1], [10] Should a feasible solution not be found, retries are possible, e.g., by changing the parameterization of the algorithm or the properties of the task set. At runtime, a very simple runtime dispatcher executes the decisions represented in the table, i.e., which (portion of a) task to execute next. Typically, a minimum granularity of time is assumed for the invocations of the runtime scheduler, so called *slots*.

II. Online Scheduling

In event triggered systems, events invoke an online scheduler, which takes a decision based on a set of pre defined rules, e.g., represented as priorities. An offline schedulability test can be used to show that, if a set of rules is applied to a given task set at runtime, all tasks will meet their deadlines. Major representative lines of such algorithms are based on fixed priorities, e.g, rate monotonic or dynamic priorities.

III. Fundamentally Different?

Given the different assumptions, approaches, and properties, offline and online scheduling could be perceived as very different, or even opposing paradigms. We will take a closer look.

The ET *real-time scheduling process* takes sets of tasks with timing constraints and performs a test if these constraints can be met if a given algorithm is used at runtime. The algorithm may take properties of tasks, notably priority or deadline, as input, or determine them as directives, artifacts for the online scheduling algorithm Then, the task properties, "priority" are separated from the importance of a task, "deadline" from the timing constraint. Rather, they both serve only to direct the online scheduling algorithm to execute the proper rules for schedulability.

The scheduling table of offline scheduling provides a "proof by construction" that all timing constraints will be met. In contrast to a proof that no timing constraints could be violated in any situation, an offline approach only needs to show that one situation exists, i.e., the scheduling table, in which the timing constraints are met: instead of a "for all" proof, considering even situations which may never occur during the runtime of a system, a "there exists one" suffices.

Thus, the process follows the steps: task set with timing constraints – schedulability test and determination of rules (e.g., via directives priority or deadline) – execution of rules by runtime scheduler – timing constraints met.

Looking closer, we can see that TT real-time scheduling work in the same way. Instead of an definition of rules, e.g., "earliest deadline first", the decisions on which task to execute are represented in the scheduling table, "schedule next task as given table".

While TT scheduling has to assume a *periodic world* and ET provides *flexibility* for tasks with not fully known parameters, e.g., aperiodic, the difference concerns mostly runtime execution without guarantees. When offline guarantees are required, task parameters have to be known offline: without worst case execution, period or maximum arrival frequency, offline guarantees cannot be given, independent of the scheduling paradigm used.

Hence, we can conclude that the terms "offline" and "online" scheduling cannot be seen as disjoint in general. Real-time scheduling requires offline guarantees, which require assumptions about online behavior at design time. At runtime, both offline and online execute according to some (explicitly or implicitly) defined rules, which guarantee feasibility. The question "offline" vs. "online" is thus less black and white, but more about how much of the decision process is Thus, both offline and online are based on a substantial offline part. The question is then where to set the tradeoff between determinism - all decisions offline - and flexibility - some decisions online.

IV. Conclusion and Question

Mixing system design issues and actual scheduling of TT and ET, offline and online scheduling appear very different. Separating activation paradigm, i.e., when is the scheduler invoked, from the actual scheduling reveals a different situation: the essential steps of (i) offline analysis and feasibility test, either via proof that no infeasible situation can occur or by constructing a complete schedule, (ii) determination of scheduling rules, whether in explicit forms, e.g., EDF, or implicit as scheduling table, and (iii) runtime execution according to these rules, are the same, albeit in different forms.

The difference between offline and online scheduling appears to be as much about asking the right question and looking at the right level as it is about where to draw the line.

References

[1] T. F. Abdelzaher and K. G. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Trans. Parallel Distributed Systems*, July 1999.

[2] Alan Burns. Programming real-time systems. In *ECRTS 04 - 16th EUROMICRO Conference on Real-Time Systems*, Catania, Sicily, July 2004.

[3] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Austria, Apr. 1994.

[4] G. Fohler and K. Ramamritham. Static scheduling of pipelined periodic tasks in distributed real-time systems. In *Proceedings of the 8th Euromicro workshop on Real-Time Systems*, June 1997.

[5] C.-J. Hou and K.G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *IEEE Proceedings of the 13th IEEE Real-Time Systems Symposium*, pages 146–155, Dec. 1992.

[6] Jan Jonsson and Kang G. Shin. A parametrized branch-and-bound strategy for scheduling precedence-constrained tasks on a multiprocessor system. In *ICPP*, pages 158–165, 1997.

[7] Hermann Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Springer, 2011.

[8] Krithi Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Trans. Parallel Distrib. Syst.*

[9] Paulo Verissimo. Fundamental questions in the ET vs. TT debate? please look elsewhere. In *Next-TTA Workshop on ET-TT Integration*, Grenoble, France, October 2002.

[10] P. Šůcha and Z. Hanzálek. Scheduling with start time realated deadlines. In *IEEE International Symposium on Computer Aided Control Systems Design*, 2004.

A Generalized View on Beneficial Task Sortings for Partitioned RMS Task Allocation on Multiprocessors

Dirk Müller, Matthias Werner
Operating Systems Group
Chemnitz University of Technology
D-09111 Chemnitz, Germany
Email: {dirkm,mwerner}@cs.tu-chemnitz.de

***Abstract*—Partitioned scheduling algorithms are typically based on Earliest Deadline First (EDF) or Rate-Monotonic Scheduling (RMS). Often, heuristics like First Fit are applied for task allocation to processors. The performance of the heuristics can be improved by sorting tasks in a preprocessing step. While for partitioned EDF, Decreasing Utilization is the approved sequence, there is a variety of sort sequences to be considered for partitioned RMS. The reason is that periods have to be taken into account. Close-to-integer-ratio periods allow for a higher utilization bound than in the general case as all ratios being integer means a simply periodic task set with utilization bound 100% on a uniprocessor. The open problem is a systematization and generalization on proposed sort sequences including the question for the best one.**

I. Introduction

Partitioned scheduling on a multiprocessor is characterized by a static mapping of n tasks with worst-case execution times (WCETs) e_i and periods p_i to processors. The deadlines shall be implicitly given, thus, $\forall i : d_i = p_i$. On the processors themselves, widely approved uniprocessor scheduling algorithms can be used. Hence, the core problem of partitioned scheduling is task allocation.

Since the problem is closely related to bin packing, it is NP-hard. The exploding number of possibilities makes an exhaustive search impractical or even impossible for greater numbers of processors and tasks.

Thus, the use of heuristics like Next Fit (NF), First Fit (FF) and Best Fit (BF) is a good compromise. Initially, they are thought as online algorithms where computation time is crucial. Purely applied, BF yields the best performance, but there is only a marginal difference to FF. Thus, FF is considered most suitable due to its lower average calculation effort [8]. Worst Fit as a fourth approach aims at load balancing and reduction of energy consumption. Balancing is reasonable because of a supralinear influence of frequency on power consumption [2].

II. Sorting as Preprocessing

A clever refinement of the heuristics is the inclusion of a *presorting*. Only for offline scheduling, such a preprocessing is possible. But the online situation for which the above mentioned heuristics where originally designed can be emulated by presenting the tasks one after another to the scheduler. Note that due to the exponential explosion of cases, the avoidance of an exhaustive search by the heuristics is still reasonable for offline scheduling. Thus, in the offline case, there is an additional degree of freedom: the sequence of the tasks in which they are presented to the (online) scheduler. So, the choice of a good sorting criterion is a naturally arising question. Traditionally, Decreasing Utilization is favored. This is reasonable for partitioned EDF [8] where period values have no influence on the fitting of tasks. Period-aware FF, a presorting according to increasing base periods, separating harmonic chains and reducing scheduling overhead, was suggested in [10] for cluster scheduling.

The situation is more complex for rate-monotonic scheduling where periods have an impact on schedulability. Here, spare utilization is determined not only by the utilizations, but also by the periods of the already allocated tasks and the new task. Two relevant contributions regarding partitioned RMS are Rate-Monotonic Small-Tasks (RMST) by Burchard *et al.* [3] using increasing S values and R-BOUND-MP by Lauzac *et al.* [7] using increasing transformed periods as sorting criterion.

RMST reorganizes the list of tasks in order to form clusters of tasks with well-fitting periods. The degree of fitting is quantified by the maximum absolute difference among the tasks' so-called S values. A task's S value is the decimal fraction of the binary logarithm of its period. Then, tasks are allocated to processors using NF and a uniprocessor scheduling criterion as well based on the S values.

R-BOUND-MP reorders tasks after a transformation of all task sets to the same (binary) order of magnitude. This scaling operation uses the binary logarithm of the ratio between maximum and own period. Thus, clusters of tasks with well-fitting periods are formed. Then, FF and the R-BOUND uniprocessor criterion based on the ratio between maximum and minimum period is applied for allocation of tasks to processors. Note that R-BOUND-MP served as a basis for the breakthrough algorithm R-BOUND-MP-NFR (Next Fit Ring) with 50% utilization bound [1]. The modification there is the substitution

of FF by NFR. The difference compared to NF is the *second chance* of allocating a task to the first processor when not succeeding allocating it to the last available processor. Note that this additional policy breaks the principle that closed bins are never touched again which was typical for classical NF.

Both sortings aim at exploiting period similarity for task allocation. Their performance in terms of success rate for synthetic task sets is similar, cf. [4] and [7]. Is this performance similarity by accident or does it have structural reasons? We claim that the small advantage of R-BOUND-MP compared to RMST detected in [7] is mainly due to the better uniprocessor criterion. Substituting the simplified form used in original RMST by an advanced one reverses the situation as shown in [9]. This underlines the impact of the uniprocessor criterion.

Interestingly, considering decreasing utilization or period similarity alone as sorting criterion comes with a limited potential of success. Attempts to unify them often apply a two-stage sorting: first decreasing utilization and then period similarity. Examples are Rate-Monotonic General-Tasks (RMGT) [3] with just two utilization classes and a threshold of $\frac{1}{3}$, and k-Rate-Monotonic-Matching (k-RMM) [5] with three main utilization classes (rough thresholds $\frac{1}{3}$ and $\frac{1}{2}$) and k subclasses where $k = \lfloor\sqrt{n}\rfloor$ turned out to be a good choice [5]. Note that this square-root choice corresponds to the default number of bins in the histogram macros of common spreadsheets. The two-stage sorting with decreasing utilization as the first criterion might be reasonable for partitioned RMS since the comprehension of period values beneficial for RMS can be regarded as a refinement.

III. Generalization

How can these different approaches be generalized and sorted into a common scheme? First, we will show that the sorting sequences [3] [7] differ just by an *index shift*. The S values shaping the Burchard sequence [3] are set by (1) to the decimal fraction of the binary logarithm of the period.

$$S_i := \log_2 p_i - \lfloor \log_2 p_i \rfloor \tag{1}$$

Lauzac *et al.* first transform the task set with a procedure *ScaleTaskSet* [7] such that all pairwise period ratios are less than 2, see (2). Then, increasing (transformed) periods give the sequence. Note that there is a mistake by an oversight in [7], flooring and logarithmization have to be exchanged.

$$p_i{}' := p_i 2^{\lfloor \log_2 (p_{\max}/p_i) \rfloor} \tag{2}$$

Since logarithmization with a base greater than 1 is strictly increasing, it does not change the order. We obtain (3) by performing a $\log_2$ operation on both LHS and RHS of (2) which is an equivalent transformation.

$$\log_2 p_i{}' := \log_2 p_i + \lfloor \log_2 p_{\max} - \log_2 p_i \rfloor \tag{3}$$

Here, $p_{\max}$ denotes the maximum period in the task set. This period serves as an offset just yielding an index shift. This index shift is arbitrary, there is no distinguished one. Thus, period similarity is a *circular*, not a linear similarity relationship. A further development of RMST should consider different index offset values. Is this circular relationship of period similarity the reason for the success of the NFR heuristics in R-BOUND-MP-NFR [1]?

This close relationship among independently suggested approaches shows the potential for generalization and integration of the other mentioned algorithms. We see this as a relevant problem which we want to solve in the future.

The theoretical consideration given above has to be extended and refined. Additionally, a case study on synthetic task sets will be performed in order to verify the hypothesis of a close relationship between RMST and R-BOUND-MP.

IV. Summary

Summing up, we question whether there exists an optimal sorting sequence for the preprocessing step in partitioned RMS heuristics as it is Decreasing Utilization for partitioned EDF. This shall include a systematization and generalization on proposed sort sequences.

Although, according to Kato, "[..] sorting the tasks does not dominate a schedulable utilization very much for the case in which the utilization of every individual task is less than 50%." [6] the problem is of paramount interest from a theoretical point of view. This includes the consideration of one-dimensional sorting criteria as well as two-stage sortings which might be more powerful. Next, both worst and average case considerations shall be performed.

References

[1] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%," in *Real-Time Systems, 2003. Proc. 15th Euromicro Conf.*, Jul 2003, pp. 33–40.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-aware scheduling for periodic real-time tasks," *Computers, IEEE Transactions on*, vol. 53, no. 5, pp. 584 – 600, may 2004.

[3] A. Burchard, J. Liebeherr, Y. Oh, and S. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1429–1442, Dec 1995.

[4] L. E. Jackson and G. N. Rouskas, "Deterministic preemptive scheduling of real-time tasks," *Computer*, vol. 35, pp. 72–79, May 2002. [Online]. Available: http://dx.doi.org/10.1109/MC.2002.999778

[5] A. Karrenbauer and T. Rothvoß, "A 3/2-approximation algorithm for rate-monotonic multiprocessor scheduling of implicit-deadline tasks," in *Proc. of the 8th int'l conf. on approximation and online algorithms*, ser. WAOA'10. Berlin: Springer, 2011, pp. 166–177. [Online]. Available: http://portal.acm.org/citation.cfm?id=1946240.1946255

[6] S. Kato, "Real-time scheduling of periodic and aperiodic tasks on multiprocessor systems," Ph.D. dissertation, Keio University, 2008.

[7] S. Lauzac, R. Melhem, and D. Mossé, "An efficient RMS admission control and its application to multiprocessor scheduling," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proc. of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, Apr. 1998, pp. 511 –518.

[8] I. Lupu, P. Courbin, L. George, and J. Goossens, "Multi-criteria evaluation of partitioning schemes for real-time systems," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, sept. 2010, pp. 1–8.

[9] D. Müller, "Accelerated simply periodic task sets for RM scheduling," in *Proc. of Embedded Real Time Software and Systems (ERTS²)*, Toulouse, 2010, p. 46. [Online]. Available: http://www.erts2010.org/Site/0ANDGY78/Fichier/PAPIERS%20ERTS%202010%202/ERTS2010_0140_final.pdf

[10] X. Qi, D. Zhu, and H. Aydin, "A study of utilization bound and run-time overhead for cluster scheduling in multiprocessor real-time systems," in *Proc. of the 2010 IEEE 16th Int'l Conf. on Embedded and Real-Time Computing Systems and Applications*, ser. RTCSA '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 3–12.

Non-Preemptive Fixed-Priority Uniprocessor Scheduling where the Execution Time of a Job Depends on the Scheduling of Jobs that Executed Before it

Björn Andersson, Dionisio de Niz and Sagar Chaki
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
{baandersson, dionisio, chaki}@sei.cmu.edu

Abstract— **We consider non-preemptive fixed-priority scheduling of a set of constrained-deadline sporadic tasks on a single processor. We assume that the execution time of a job J depends on the sequence of jobs executed before J, that is, the execution time of a job of a task is not a constant. We raise the following two open problems (i) given a priority assignment, can the response time of a task be computed in pseudo-polynomial time? and (ii) how to create an optimal priority-assignment scheme?**

I. MOTIVATION

With today's processors, the execution time of a task is heavily dependent on whether a memory operation (load/store) results in a cache hit or a cache miss. Even with non-preemptive scheduling and even with a single processor, the execution time of a job of a task depends on the scheduling of other tasks. As an illustration, consider tasks τ_1, τ_2 and τ_3 which are scheduled non-preemptively on a single processor. Jobs of task τ_1 start their execution by reading a variable x, where x is an array. Ditto for the jobs of task τ_2: jobs of task τ_2 start their execution by reading a variable x. However, jobs of task τ_3 never access variable x. For each of the tasks τ_1 and τ_2, it holds that its jobs have the execution time 5 milliseconds if variable x was not in the cache. If a job of task τ_2 executed immediately after a job of task τ_1, then τ_1's job will experience a cache miss when referencing the variable x but we may be able to prove than when the job of task τ_2 references variable x, it results in cache hits when τ_2 references x and hence the execution time of τ_2's job becomes 4 milliseconds.

Therefore, we need a scheduling theory which takes into account the fact that the execution time of a job may depend on which jobs executed just before it. Unfortunately, the current research literature offers no such scheduling theory.

II. MODEL

Task and platform characterization. We consider a system comprising a single processor and a software system comprising a task set τ composed of n constrained-deadline sporadic tasks. A task $\tau_i \in \tau$ is characterized by integers D_i and T_i with the interpretation that the task generates a (potentially infinite) sequence of jobs where the arrival times of jobs by τ_i are separated by at least T_i time units and a job of task τ_i must finish its execution within D_i time units after its arrival.

The execution time of a job depends on the job executing before it, and we therefore define the following concepts for task τ_i. The symbol $nhistories_i$ is an integer greater than or equal to one. $historylength_i^h$ is an integer greater than or equal to one and it is defined for $1 \le h \le nhistories_i$. The symbol $historyitem_i^{h,k}$ is an integer in $\{1,2,3,\ldots,n\}$ and it is defined for $1 \le h \le nhistories_i$ and $1 \le k \le historylength_i^h$.

We say that the for job J generated by task τ_i, the execution time C_i^h is *historyallowed* if it holds for the $historylength_i^h$ jobs that executed before J that for each $j \in \{1,2,\ldots,historylength_i^h\}$, the j^{th} job before J is the generated by the task with index $historyitem_i^{h,j}$. The execution time of a job is the minimum among all its *historyallowed* execution times. Note that C_i^h is *historyallowed* if $historylength_i^h=0$. We assume that for each task τ_i there is one h such that $historylength_i^h=0$.

Figure 1a shows an example task set in this model.

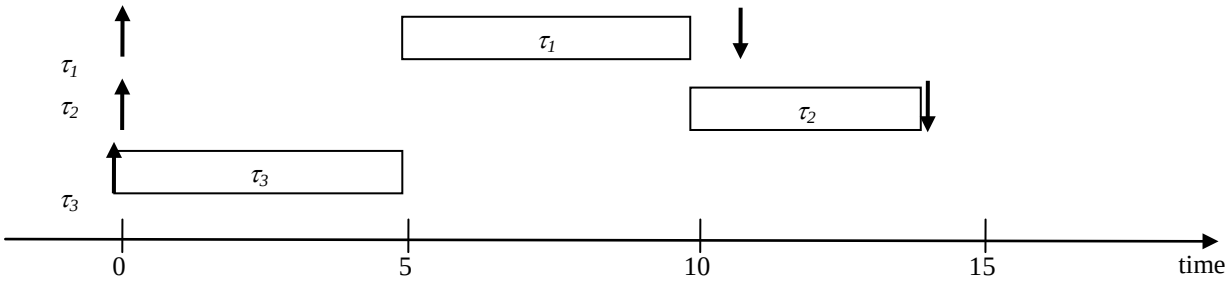
Scheduling. We assume that each task τ_i is assigned a priority $prio_i$ and each job generated by task τ_i is given the priority of the task that generated the job. We say that a job J is *eligible for execution* at time t, if (i) job J arrives at t or earlier and (ii) job J finishes execution later than t.

We assume non-preemptive scheduling, that is, if a job has started to execute then it will continue to execute until it finishes. When a job finishes, the job selected for execution is the one with the highest priority among the jobs that are eligible for execution at that time. Figure 1(b) and Figure 1(c) show examples of schedules.

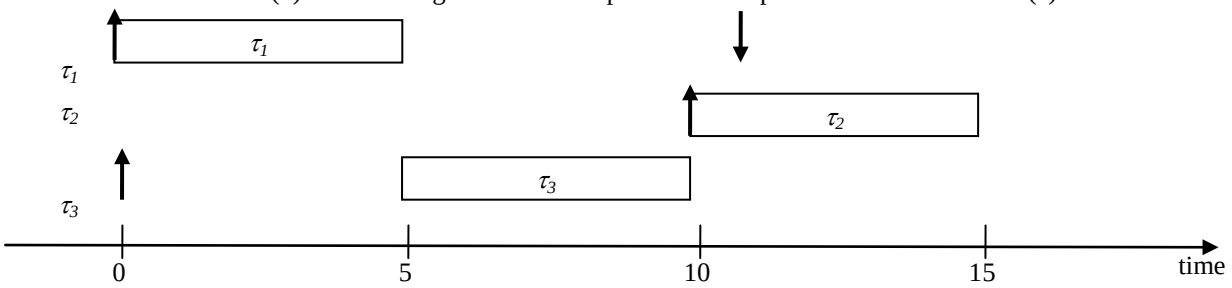
n=3,

T_1=50, D_1=11,	T_2=150, D_2=14,	T_3=500, D_3=500,
$nhistories_1$=2	$nhistories_2$=2	$nhistories_3$=1
C_1^1=5, $historylength_1^1$=0	C_2^1=5, $historylength_2^1$=0	C_3^1=5, $historylength_3^1$=0
C_1^2=4, $historylength_1^2$=1, $historyitem_1^{2,1}$=2	C_2^2=4, $historylength_2^2$=1, $historyitem_2^{2,1}$=1	

(a) An example of a task set. This task set models that task τ_1 and τ_2 share some variables and therefore executing one of them just before the other reduces the execution time of the other.



(b) A schedule generated for a specific arrival pattern for the task set in (a).



(c) A schedule generated for another specific arrival pattern for the task set in (a).

Figure 1. An example of a task set and two examples of schedules that can be generated for different arrival patterns.

Response time and schedulability. The response time of a job is the time that the job finishes execution minus the arrival time of the job. The response time of a task τ_i (denoted R_i) is the maximum response time that a job of τ_i can experience. We say that a task set is *schedulable* with respect to priority assignment P if $\forall i$: $R_i \leq D_i$. We say that a task set is *non-preemptive fixed-priority feasible* if there exists a priority assignment such that the task set is schedulable with respect to this priority assignment. We say that a priority-assignment scheme A is *optimal* if for each task set that is non-preemptive fixed-priority feasible, the task set is schedulable with respect to the priority given by the priority-assignment scheme A.

Note that in Figure 1(b), the job of task τ_2 has execution time of four time units because it executes after a job of task τ_1. Hence the job of task τ_2 meets its deadline. Classical non-preemptive analysis however, which does not consider that the execution time of a job depends on the job that executes before it, would calculate an upper bound on the response time being one time unit longer than the one in the example in Figure 1(b) and hence classical non-preemptive analysis would deem the task set in Figure 1(a) unschedulable.

Figure 1(c) shows an example of a schedule for another arrival pattern. Note here that the job by τ_2 does not execute directly after a job of task τ_1 and hence the execution time of the job of task τ_2 is five, that is, one time unit more than it was in Figure 1(b).

III. Open Problem Formulation

We propose the following two open problems:

OP1. Is it possible to create an algorithm, with pseudo-polynomial time-complexity, which computes R_i?

OP2. How to create an optimal priority assignment scheme.

We believe these two problems are interesting because both of them have (affirmative/positive) answers/solutions for the case that the execution time of a job does not depend on its history. But for our model they are unresolved.

Acknowledgment

We are grateful to discussions with Russell Kegley (Lockheed Martin), Brian Dougherty (Virginia Tech) and Jules White (Virginia Tech) who brought to our attention this problem.

The Explicit Preemption Placement Problem for Real-Time Conditional Code

Marko Bertogna[1] and Nathan Fisher[2]

[1]Scuola Superiore Sant'Anna, Pisa, Italy, marko.bertogna@sssup.it
[2]Department of Computer Science, Wayne State University, Detroit, Michigan, USA, fishern@cs.wayne.edu

I. Introduction

In real-time worst-case execution time (WCET) analysis, an upper bound is calculated, for each job in the system, on the total aggregate amount of execution required to successfully complete the job. Real-time schedulability analysis has traditionally used the estimates determined from WCET analysis to determine whether every job in a system can be completed by its deadline. Thus, the effectiveness of the resulting schedulability analysis hinges upon the precision of WCET estimates. Unfortunately, many of scheduler properties that simplify schedulability analysis often introduce pessimism into WCET analysis. For example, the oft-assumed property that jobs are arbitrarily preemptible leads to significant increase in the WCET estimates, as the analysis must assume that a preemption occurs often and the overhead of such preemption (due to context switch time and cache effects) must be added to the estimate. Furthermore, most real-time scheduling algorithms and associated schedulability analysis do not take the heterogeneous "cost" of preemption overhead into account when making scheduling decisions. For instance, preemption of a job may cause cache lines to be invalidated that are needed in subsequent instructions; the memory access pattern of nearby instructions will greatly influence the cost of the preemption due to such invalidations. Thus, a better strategy may be to delay the preemption of a job that is executing instructions with a degree of spatial or temporal locality (in terms of memory access) until it reaches instructions with a lower level of memory locality.

Very recently, Bertogna et al. [1] (in ECRTS 2011) proposed such an approach that explicitly and efficiently determines (prior to runtime) the optimal choice of *explicit preemption points* (EPPs) in a job's code that minimize the preemption overhead while ensuring that system schedulability is not affected due to increased non-preemptivity. However, their proposed approach only deals with linear (non-branching) code and cannot handle jobs with control flow such as conditional statements (e.g., if-then-else statements) and loops. In this abstract, we propose specific open problems towards extending the EPP approach of Bertogna et al. [1] for handling general conditional code. Furthermore, our objective is to obtain a solution that retains the efficient running time of the non-branching version of the problem. We believe that such extensions are absolutely necessary for the EPP approach to be widely applicable and useful to a real-time system designer.

II. Model

We refer to the problem of determining the optimal choice of EPPs for a program (i.e., job) as the *explicit preemption placement* problem. To model the explicit preemption placement problem, we assume that a program $\mathcal{P}$ has been divided into a set of non-preemptive basic blocks (BBs). (Currently, [1] assumes that any conditional code is entirely contained within a single BB). A directed graph $G_{\mathcal{P}} = (V, E)$ describes the control flow of $\mathcal{P}$. Each vertex $v \in V$ represents a BB in $\mathcal{P}$. An edge $(u, v) \in E \subseteq V \times V$ means that the execution of BB u immediately precedes the execution of BB v in some execution path of $\mathcal{P}$, and that a preemption is permitted between the two BBs, i.e., E is the set of possible EPPs. We assume that there is a special vertex s that indicates the initial BB of the $\mathcal{P}$. Similarly, there is a special vertex z that is the terminating BB of $\mathcal{P}$, i.e., that has no successor BB. A path p is an ordered set of consecutive vertices, such that each vertex in p has an edge from its predecessor. Let $\mathtt{paths}$ be the set of possible execution paths from the initial BB s to the terminating BB z.

For the purposes of quantifying the preemption overhead of selecting an EPP, we assume that a function $\xi : E \mapsto \mathbb{R}_{\geq 0}$ is given. Similarly, the WCET of a BB is given by a function $C : V \mapsto \mathbb{R}_{\geq 0}$. Finally, since the selection of EPPs will create non-preemptible regions in $\mathcal{P}$, the schedulability of the system is affected by a choice of EPPs. Thus, we will assume that a constant Q is determined which quantifies the maximum duration of any non-preemptive region in $\mathcal{P}$.

Note that since every structured program can be expressed as a combination of sequential instructions, conditional branches and loops, the adopted model is general enough to express every real-time task implemented with a structured programming language.

III. Open Problems

Given the above model, our goal is to find a selection of EPPs that minimize the WCET of $\mathcal{P}$. More formally, our main open problem is:

Given $G_{\mathcal{P}}$ and associated functions ξ and C, find $S \subseteq E$ that minimizes

$$\max_{p \in \mathsf{paths}} \left\{ \sum_{u \in p} C(u) + \sum_{\substack{u,v \in p \\ (u,v) \in S}} \xi(u,v) \right\} \quad (1)$$

subject to the constraint that, for each path in paths and for any two $u, v \in p$, if the total execution of BBs from u to v in path p is greater than Q then some edge of p must be in S.

To solve the above general problem, we must answer the following subproblems:

SP1 *Given conditional, non-looping code, can a solution be determined in a time polynomial in the number of EPPs?* Systems without loops, or with loops entirely contained inside a BB, can be modeled using Directed Acyclic Graphs (DAGs), simplifying the problem. However, no optimal method is known even for this simplified problem, due to the presence of conditional branches. In fact, the selection of EPPs for one path of the DAG might be conflicting with the selection for other paths that share some vertex.

SP2 *How do loops affect the analysis?* As it is common in the timing analysis domain, we are only interested in loops that have a deterministic number of iterations.

REFERENCES

[1] M. BERTOGNA, O. XHANI, M. MARINONI, F. ESPOSITO, AND G. BUTTAZZO. Optimal Selection of Preemption Points to Minimize Preemption Overhead. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Porto, Portugal, July 2011. IEEE Computer Society Press.

Probabilistic Sensitivity Analysis

Luca Santinelli*, Liliana Cucu-Grosjean* and Laurent George♯
* INRIA Nancy Grand Est, Nancy, France
♯ University of Paris 12, Paris, France
luca.santinelli@inria.fr, liliana.cucu@inria.fr, lgeorge@ieee.org

I. Introduction

Probabilistic real-time systems and probabilistic real-time analysis became a common practice in the real-time community, [3]. Probabilistic approaches are promising because they answer questions or provide solutions that cannot be addressed in a deterministic manner such as the distributions for the response times. Moreover, they consider models that are more realistic, for instance regarding the message activation patterns or the expression of soft real-time constraints.

Papers related to our work had equally used the words *stochastic analysis* [8], *probabilistic analysis* [11], *statistical analysis* [1] and *real-time queuing theory* [9]. Since the paper of Diaz et al. [5], the term *stochastic analysis* of real-time systems has been used regularly by the community regardless of the approach (probabilistic or statistical). While the word *stochastic*, is often associated with unpredicted behavior, we make use of the word *probabilistic* in order to indicate that the work is based on the theory of probability. Moreover by *probabilistic real-time system* we mean a real-time system with at least one parameter defined by a random variable, [10], [7], [12], [4].

II. Model

We consider a real-time system Γ composed of n tasks where each task τ_i is characterized by three parameters $(\mathcal{C}_i, D_i, \mathcal{T}_i)$. $\mathcal{C}_i$ is the execution time described by a random variable with a known probability function $f_{\mathcal{C}_i}(\cdot)$ with $f_{\mathcal{C}_i}(c) = P(\mathcal{C}_i = c)$, $\mathcal{T}_i$ is the random variable describing the task period with a known probability function denoted by $f_{\mathcal{T}_i}(\cdot)$ with $f_{\mathcal{T}_i}(T) = P(\mathcal{T}_i = T)$; $D_i \leq min\{T_i\}$ is the relative deadline of the task. The discrete random variables of the execution time can be written as follows

$$\mathcal{C}_i = \begin{pmatrix} C_i^{min} = C_i^0 & C_i^1 & \cdots & C_i^{max} = C_i^m \\ f_{\mathcal{C}_i}(C_i^{min}) & f_{\mathcal{C}_i}(C_i^1) & \cdots & f_{\mathcal{C}_i}(C_i^{max}) \end{pmatrix},$$

with m the possible execution times for the tasks, each one with a probability of happening associated, as already introduced in [5]. For the periods, the random variable can be

$$\mathcal{T}_i = \begin{pmatrix} T_i^{min} = T_i^0 & T_i^1 & \cdots & T_i^{max} = T_i^r \\ f_{\mathcal{T}_i}(T_i^{min}) & f_{\mathcal{T}_i}(T_i^1) & \cdots & f_{\mathcal{T}_i}(T_i^{max}) \end{pmatrix},$$

with r the possible periods for the task instances. On top of such a model, it has been developed the analysis based on the response time computation, as in [5]. As a result, the distribution of the response time is obtained.

Example 2.1: Let τ_i be a task with the execution time described by the random variable

$$\mathcal{C}_i = \begin{pmatrix} 3 & 4 & 5 & 6 & 7 \\ 0.05 & 0.1 & 0.15 & 0.4 & 0.1 \end{pmatrix} \cup \begin{pmatrix} 8 & 9 & 10 & 11 & 12 & 20 \\ 0.06 & 0.04 & 0.03 & 0.03 & 0.03 & 0.01 \end{pmatrix};$$

Figure 1 describes the cumulative distribution function of $\mathcal{C}_i$.

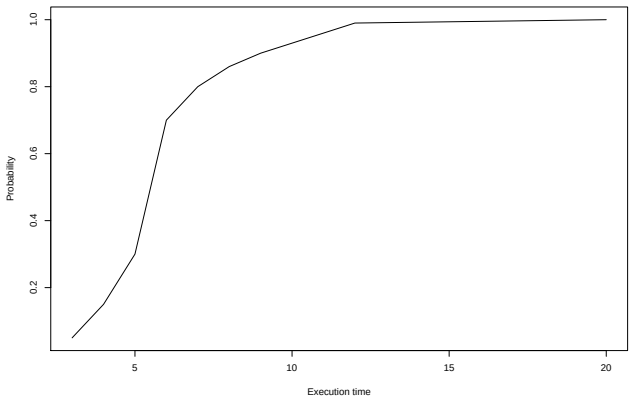


Figure 1. Cumulative distribution function of the execution time $\mathcal{C}_i$.

III. Sensitivity Analysis

The sensitivity analysis relies on the context of a new representation of the feasibility condition as a region defined in the space of those task attributes X considered as design variables. In particular, most of the works defines the schedulability region in the space of the worst-case computation times ($X = C$), referred to

as the C-space. Since the sensitivity analysis provides theory and algorithms for measuring the distance of a scheduling solution from the boundary of the feasibility region, such a distance, expressed in the C-space is an exact measure of the available slack or, conversely, of the correcting actions that must be taken on the computation times to make the system schedulable.

The sensitivity analysis has been exploited in both cases of fixed priority and dynamic priority scheduling, [2], [6], based on the classical deterministic model of real-time systems. Instead, the probabilistic model of tasks has to be applied on top of the results and the conditions derived by the classical sensitivity analysis allowing to further conclude about the system and its requirements. For example, the probabilistic model could fractionate the feasibility region into

- *total* feasibility regions: where the feasibility is guaranteed 100%, e.g. all the job instances and 100% of the time the deadline are guaranteed.
- *partial* feasibility regions: where the feasibility is not guaranteed 100%, but there are probability thresholds associated; i.e. a feasibility region at 90% where each point inside that region represents a system configuration where at least 90% of the job instances are guaranteed to meet their deadlines.

The probabilistic characterization will bring flexibility into the sensitivity analysis allowing one to differentiate among feasible conditions. Such a flexible probabilistic sensitivity analysis could cope with particular schedulability conditions and the real-time degree asked by the system under investigation. Indeed, it could deal with both hard and soft real-time systems and all the grades in between with probability thresholds applied. Furthermore, the probabilistic schedulability analysis

- could extend the analysis to multiple execution time requirements;
- it could reduce the pessimism of deterministic analysis by facing the specific schedulability condition required;
- it could allow a more detailed feedback to the design of real-time system.

Those benefits have to be verified with an accurate investigation and an effective application of the probabilistic sensitivity analysis that has to tackle with the representation of the probabilistic model within the C-space and the possible outputs of the analysis as well as other open issues.

REFERENCES

[1] A.K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *19th IEEE Real-Time Systems Symposium*, 1998.

[2] Enrico Bini, Marco Di Natale, and Giorgio C. Buttazzo. Sensitivity analysis for fixed priority real-time systems. In *18th Euromicro Conference of Real-Time Systems (ECRTS 2006)*, 2006.

[3] A. Burns, G. Bernat, and I. Broster. A probabilistic framework for schedulability analysis. In *Third International Embedded Software Conference (EMSOFT03)*, pages 1–15, 2003.

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

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

[6] Jean-Francois Hermant and Laurent George. A c-space senstivity analysis of earliest deadline first scheduling. In *Workshop on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2007)*, 2007.

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

[8] Giordano A. Kaczynski, Lucia Lo Bello, and Thomas Nolte. Towards stochastic response-time of hierarchically scheduled real-time tasks. In *Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2006*, pages 453–456, 2006.

[9] J.P. Lehoczky. Real-time queueing theory. In *10th of the IEEE Real-Time Systems Symposium*, 1990.

[10] J.M. Lopez, J.L. Diaz, J Entrialgo, and D. Garcia. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Journal of Real-time Systems*, 40(2), 2008.

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

[12] Haibo Zeng, Marco Di Natale, Paolo Giusto, and Alberto L. Sangiovanni-Vincentelli. Stochastic analysis of distributed real-time automotive systems. *IEEE Trans. Industrial Informatics*, 5(4):388–401, 2009.

What is the meaning of preemption in utility-based real-time scheduling?

Raphael Guerra, Gerhard Fohler
Technische Universität Kaiserslautern, Germany
{guerra,fohler}@eit.uni-kl.de

I. Preliminaries

Time utility function (TUF) scheduling goes beyond the starttime-deadline notion to express tasks' temporal constraints. The goal of the scheduler is to maximize system utility under the assumption that tasks aggregate a given amount of utility to the system as a function of when they execute. Work available in the literature include the characterization of time utility functions to express applications' requirements [5], [7], and several proposals of scheduling algorithms for increased utility accrual [8], [9], [3] which differ in their supported utility functions, overhead, etc.

Most work on TUF scheduling, including all previously mentioned scheduling algorithms, do not consider preemption. Many simple embedded system do not support preemption, and some applications, like network packet scheduling, are inherently non-preemptive. However, if allowed, preemption opens up possibility for more efficient resource utilization, and being able to explore it is advantageous. We believe that the scarcity of work considering preemption reflects the lack of understanding of the impact of preemption on the utility accrual. Moreover, preemptive scheduling imposes extra challenges such as context switch overhead, which may lead to poor and inefficient resource utilization.

II. The relation between preemption and utility accrual

To the best of our knowledge, preemptive TUF scheduling has been addressed only in [6] and [2]. The work in [6] proposes a TUF preemptive scheduling algorithm assuming that the time of completion of a task defines the utility accrual to the system. A previous version of the work had focused on network packet scheduling, where this assumption holds, but the problem is inherently non-preemptive. However, as we will see later in this section, not all applications accrue utility to the system as a function of the completion time. Therefore, this model does not generalize the expression of the impact of preemption on the accrued utility.

The work in [2] proposes a preemptive TUF based scheduler which assumes that each instruction of a job's execution accrues utility to the system as a function of the moment of execution. The utility of a job is, then, the integral of the utility function within the time intervals this job executes (see figure 1). This figure depicts the schedule of a job which executes in the time intervals $[t_1, t_2]$ and $[t_3, t_4]$, and the corresponding utility accrual.

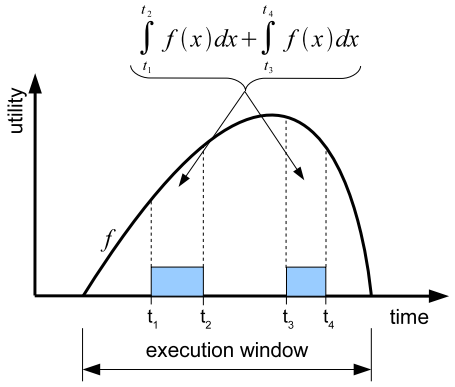


Figure 1. TUF model in [2].

We believe that this task model has some flaws in expressing the utility accrual of applications as a function of time. For example, the authors of [2] use as motivating example a tracking system which verifies whether objects cross a certain boundary and intercepts them. This system reads the coordinates of the objects from registers that are periodically updated by a sensory system, processes some data, and generates as output the interception point based on trajectory pattern and speed. The utility of this application depends on the age of the sensed position when applying the interception (the older the data, the less accurate the position of the object), and the time of the output (early or late action will fail to intercept the object). Therefore, not all instructions of the code alter the utility that the application accrues to the system.

Let us analyze the timeliness requirements of other 2 applications for a better understanding of the impact of preemption on system utility: multimedia and control. Video decoding and playout require strictly periodic frame display for maximum Perceived Quality of Video (PQV) [1]. The PQV varies as a function of the display time of each frame, and frames can be displayed only after decoded. Since buffering frames in advance is not an option for high consumer electronics [4], frames must be displayed before the next frame starts being decoded. In this application, the utility relates to the PQV, which depends only on the decoding completion (moment of frame display), and hence, is independent of preemptions.

The basic timing parameters of control tasks are shown in figure 2. Control tasks are released (e.g., inserted into the ready queue of the real-time operating system) periodically at times r_k, and $r_{k+1} - r_k = p$, where p is the period of the controller. Due to preemption from other

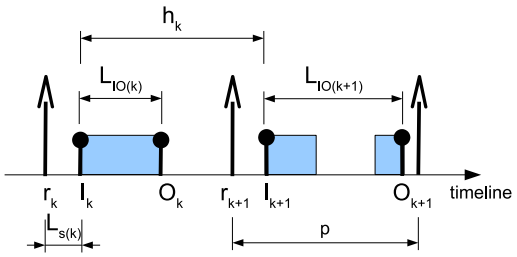


Figure 2. Control timing constraints.

tasks in the system, the actual start of the task may be delayed for some time $L_{s(k)}$. This is called the *sampling latency* of the controller. A dynamic scheduling policy will introduce variations in this sampling latency across intervals. These variations are called *sampling jitter*. The maximum sampling jitter is quantified by the difference between the maximum and minimum sampling latencies in all task instances, thus $J_s^{MAX} = L_s^{MAX} - L_s^{MIN}$.

The *sampling interval latency* h_k is the interval of time between two consecutive samplings I_k, thus $h_k = I_{k+1} - I_k$. The *nominal sampling interval* is such that $h_k = p$. Jitter in the sampling latency will of course also introduce jitter in the sampling interval latency, called *sampling interval jitter*. The maximum sampling interval jitter is $J_h^{MAX} = L_s^{MAX} + L_s^{MIN}$.

After some computation time and possibly further preemption from other tasks, the controller will actuate the control signal (or control output) at time O_k. The delay from the sampling to the actuation is the input-output latency $L_{IO(k)} = O_k - I_k$. Varying execution times or task scheduling preemptions will introduce variations in this interval. The maximum *input-output jitter* is quantified by the difference between the maximum and minimum input-output sampling latencies in all task instances, thus $J_{IO}^{MAX} = L_{IO}^{MAX} - L_{IO}^{MIN}$.

Basic control theory assumes all latencies to be zero, which also implies that all jitters are zero, for optimum control performance (maximum utility). Latencies and jitters are tolerable, albeit at lower utility. In this application, preemptions may impact on the utility accrual due to input-output latency.

III. How to express the relation between preemption and utility accrual in task models?

One of the open problems is which instructions of a task contribute to the utility accrual. The examples and observations of the previous section lead us to conclude that the utility that an application accrues to the system varies as a function of the moment of I/O operations. The internal computational state of an application is not visible to the system, and hence, should not alter the utility accrual.

Another problem is the task model abstraction which is necessary to express the utility accrual of applications. We believe that a potential solution is to extend task models with the information of which instructions of a task accrue utility to the system. Such an extension to real-time task models provides for the generalization of the task models proposed in [2] and [6]. A task may express that only one instant of the execution defines the utility accrual, like the moment completion as proposed in [6], and that every instruction accrues utility, as the model in [2]. We wonder which other application examples back up our proposal, and whether a designer can define all points of utility accrual for every application.

References

[1] M. Claypool and J. Tanner, "The effects of jitter on the perceptual quality of video," *Proc. ACM Multimedia '99(Part 2)*, pp. 115–118, 1999. [Online]. Available: citeseer.ist.psu.edu/claypool99effect.html

[2] L. Farzinvash and M. Kargahi, "A scheduling algorithm for execution-instant sensitive real-time systems," in *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09. 15th IEEE International Conference on*, 2009, pp. 511 –518.

[3] R. Guerra and G. Fohler, "A gravitational task model with arbitrary anchor points for target sensitive real-time applications," *Real-Time Syst.*, vol. 43, no. 1, pp. 93–115, 2009.

[4] D. Isovic, G. Fohler, and L. F. Steffens, "Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions," in *15th Euromicro Conference on Real-time Systems (ECRTS 03)*, Porto, Portugal, July 2003.

[5] E. D. Jensen, "Asynchronous decentralized real-time computer systems," in *Real-Time Computing*, ser. the NATO Advanced Study Institute, W. A. Halang and A. D. Stoyenko, Eds. Springer Verlag, October 1992.

[6] P. Li, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Trans. Comput.*, vol. 55, no. 4, pp. 454–469, 2006, member-Haisang Wu and Senior Member-Binoy Ravindran and Member-E. Douglas Jensen.

[7] D. Prasad, A. Burns, and M. Atkins, "The valid use of utility in adaptive real-time systems," *Real-Time Syst.*, vol. 25, no. 2-3, pp. 277–296, 2003.

[8] J. Wang and B. Ravindran, "Time-utility function-driven switched Ethernet: Packet scheduling algorithm, implementation, and feasibility analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 2, pp. 119–133, 2004.

[9] H. Wu, U. Balli, B. Ravindran, and E. D. Jensen, "Utility accrual real-time scheduling under variable cost functions," in *RTCSA '05: Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 213–219.

Partitioned Scheduling of Multimode Systems on Multiprocessor Platforms: when to do the Mode Transition?

José Marinho, Gurulingesh Raravi, Vincent Nélis, Stefan M. Petters
CISTER-ISEP Research Center, Polytechnic Institute of Porto, Portugal.
{jmsm, ghri, nelis, smp}@isep.ipp.pt

Consider the problem of executing a multimode real-time system using partitioned scheduling on a multiprocessor platform. The model of computation is as follows (all technical terms are interpreted according to their usual definitions):

- **Platform:** The platform is composed of m identical processors $\{\pi_1, \ldots, \pi_m\}$.
- **System:** The system is composed of a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task is assumed to be sporadic and implicit-deadline. The n tasks are divided into k subsets $\{\tau^1, \tau^2, \ldots, \tau^k\}$, where each subset τ^j represents the tasks that have to be executed while the system is running in mode M_j. The system can run in k different operational modes. These subsets τ^j are exhaustive but not mutually exclusive, i.e., $\tau^1 \cup \tau^2 \cup \ldots \cup \tau^k = \tau$ and $\exists j, i : \tau^i \cap \tau^j \neq \emptyset$. The tasks belonging to more than one mode are called *Mode-Independent* (MI) tasks.
- **Mode Transition:** At run-time, the application is either running in one of the modes (say M_i) or it is switching from one mode (i.e., the old mode) to another (i.e., new mode). A *mode transition phase* starts with a Mode Change Request (MCR) and ends when all the old mode tasks have completed the execution of their last released job (those old mode tasks do not release new jobs after an MCR) and all the new mode tasks have been activated, i.e. they have started to release jobs. Note that a mode-independent task that belongs to both the old and new mode should not be affected by the mode change in progress. Additionally no deadlines in the system may be violated.
- **Scheduling:** We consider partitioned scheduling, i.e., the task set τ^j of each mode M^j is partitioned into m subsets $\tau^{j,1}, \tau^{j,2}, \ldots, \tau^{j,m}$. Each partition $\tau^{j,\ell}$ is statically assigned to processor π_ℓ and is scheduled by preemptive EDF.
- **Assumptions:** Tasks are not allowed to migrate between processors during the execution of any mode. Furthermore, we assume that every subset of tasks $\tau^{j,\ell}$ $(\forall j, \ell)$ is EDF-schedulable.

Open problem: For any given transition phase, what is the earliest time-instant after the MCR at which all the new mode tasks can be safely activated?

Observation 1. *In order to preserve the schedulability of the system, it might be the case that some MI tasks have to migrate from one processor to another when a mode transition is initiated.*

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
C_i	20	14	20	30	1	6
T_i	39.2	30	60	60	10	10
U_i	0.51	0.46	0.4	0.5	0.1	0.6
belongs to	τ^1	MI	MI	MI	τ^1	τ^2

Table I: A task set to illustrate the necessity of task migration across different modes.

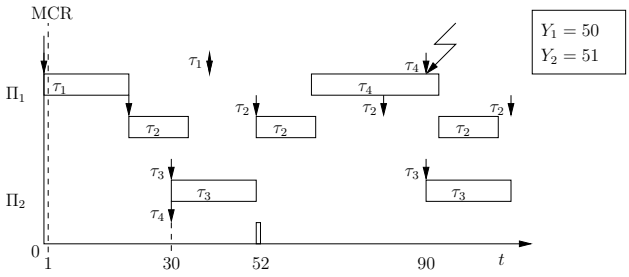


Figure 2: Schedule missing deadline with uniprocessor protocol adaptation

Consider a system with two modes M_1 and M_2, and six tasks $\{\tau_1, \ldots, \tau_6\}$. These tasks have to be scheduled on two processors π_1 and π_2, and their parameters are given in Table 1. The notation MI indicates that the task belongs to both modes M_1 and M_2. Thus we have $\tau^1 = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$ and $\tau^2 = \{\tau_2, \tau_3, \tau_4, \tau_6\}$. It can be shown that the only EDF-schedulable partitions for τ^1 is $\{\{\tau_1, \tau_2\}, \{\tau_3, \tau_4, \tau_5\}\}$. Similarly, the only EDF-schedulable partitions for τ^2 is $\{\{\tau_2, \tau_4\}, \{\tau_3, \tau_6\}\}$. That is, there is no EDF-schedulable partitions in which every MI task is assigned to the same processor in both modes M_1 and M_2, hence providing Observation 1.

Observation 2. *While it is commonly believed that uniprocessor scheduling techniques can be directly applied to partitioned multi-core scheduling, the presence of MI tasks may lead to counter-intuitive results.*

Consider the following uniprocessor result [1]: for any transition phase from mode M_i to mode M_j, the new-mode tasks can be safely activated at any time $t \geq t_{\text{MCR}} + Y_{i,j}$, where t_{MCR} is the time-instant of the last MCR and $Y_{i,j} \stackrel{\text{def}}{=} \sum_{\tau_\ell \in \tau^i} C_\ell + \sum_{\tau_\ell \in \tau^i \cap \tau^j} \left\lceil \frac{Y_{i,j}}{T_\ell} \right\rceil \times C_\ell$. A direct adaptation of the above protocol to partitioned multi-core would be: for any transition phase from mode M_i to mode M_j, the new-mode tasks can be safely activated at any time $t \geq t_{\text{MCR}} + \max_{k=1}^{m} \{Y_{i,j}^k\}$, where t_{MCR} is the

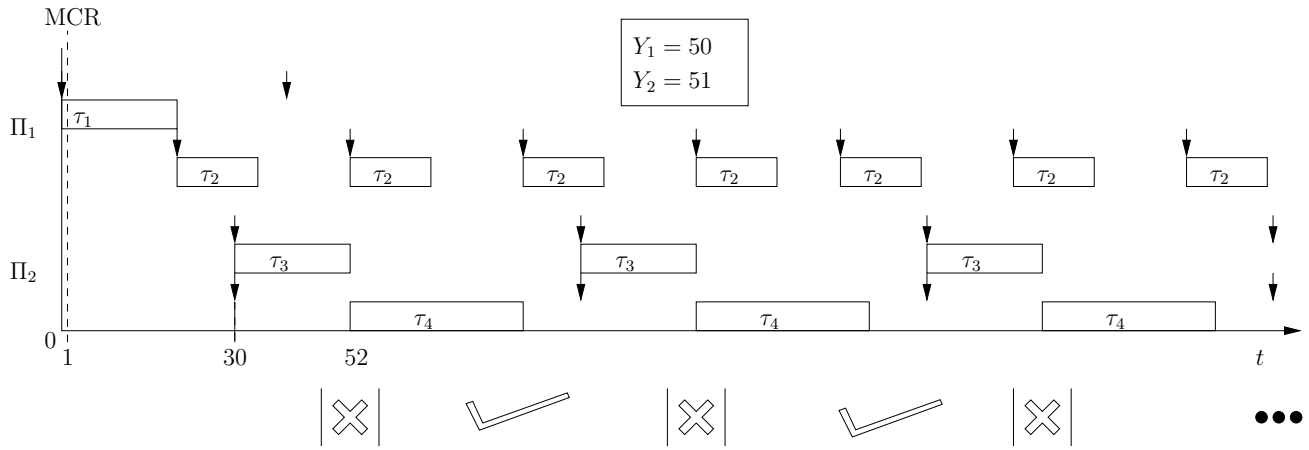


Figure 1: schedule of the two partitions with mode transition prolonged

instant of the last MCR and

$$Y_{i,j}^k \stackrel{\text{def}}{=} \sum_{\tau_\ell \in \tau^{i,k}} C_\ell + \sum_{\tau_\ell \in \tau^{i,k} \cap \tau^{j,k}} \left\lceil \frac{Y_{i,j}^k}{T_\ell} \right\rceil \times C_\ell.$$

However, the example using the system described in Figure 1 proves that this straight-forward extension of the protocol lead to deadline miss. In Figure 2 a deadline is missed at time $t = 90$ after the assumed end of the mode transition phase which completes at $t = 52$ since $t_{\text{MCR}} = 1$ and $\max_{k=1}^{m} \left\{ Y_{i,j}^k \right\} = 51$. It thus shows that waiting for the old mode workload to finish in every cores is not a sufficient condition to command the mode transition.

Observation 3. *There exist time intervals in which mode transition can be performed without missing any deadlines in the system. These intervals are separated by time intervals in which mode transition cannot be performed without missing a deadline.*

The extended schedule of the system discussed in Observation 1 is shown in Figure 1. We can observe from the schedule that the time intervals where it is safe to perform mode transition and time intervals where it is not safe to perform mode transition alternate. Hence, it is difficult to find at design time a time instant (using a uniprocessor multimode protocol) at which it is always safe to perform mode transition (as MCR is a run-time event).

Hence, from Observations 2 and 3, we can conclude that the uniprocessor multimode scheduling techniques cannot be directly applied to a multiprocessor multimode partitioned scheduling scenario.

ACKNOLEDGEMENTS

This work was supported by the RePoMuC project, ref. FCOMP-01-0124-FEDER-015050, funded by FEDER funds through COMPETE (POFC - Operational Programme Thematic Factors of Competitiveness) and by National Funds (PT) through FCT - Portuguese Foundation for Science and Technology and the RECOMP the project, funded by National Funds, through the FCT, under grant ref.- ARTEMIS/0202/2009 , as well as by the ARTEMIS Joint Undertaking, under grant agreement Grant nr. 100202.

REFERENCES

[1] Jorge Real, Alfons Crespo: Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. Real-Time Systems 26(2): 161–197 (2004).

Two-type Heterogeneous Multiprocessor Scheduling: Is there a Phase Transition?

Gurulingesh Raravi*, Björn Andersson†* and Konstantinos Bletsas*
*CISTER-ISEP Research Center, Polytechnic Institute of Porto, Portugal
†Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA
Email: *{ghri, baa, ksbs}@isep.ipp.pt, †baandersson@sei.cmu.edu

I. Introduction

Consider the problem of non-migratively scheduling a set of implicit-deadline sporadic tasks to meet all deadlines on a two-type heterogeneous multiprocessor platform.

A. System Model and Assumptions

The system is as follows:

- **Computing Platform (denoted as Π):** The computing platform consists of m processors; Of those, $m_1 \geq 1$ are of type-1, and $m_2 \geq 1$ are of type-2, i.e., $m_1 + m_2 = m$. A processor is denoted as $\pi_j \in \Pi$, where $j \in \{1, \cdots, m\}$.
- **Task Set (denoted as τ):** The task set comprises n implicit-deadline sporadic tasks (i.e., for each task, its deadline is equal to its minimum inter-arrival time). A task is denoted as $\tau_i \in \tau$, where $i \in \{1, \cdots, n\}$.
- **Utilization (denoted as U):** The utilization of a task τ_i on a processor π_j is given by u_i^j, a non-negative real number.

The following assumptions are made:

- **No job parallelism:** A job can be executing on at most one processor at any given time instant
- **Independent tasks:** The execution of jobs are independent, i.e., they neither share any resources nor have data dependency and
- **No migration:** All the jobs released by a task must execute on the same processor to which the task is assigned.

B. Phase Transition

A behavior in which a given system transitions from one state to another is known as *phase transition* behavior. During the phase transition, certain properties of the system change "drastically". In context of real-time scheduling, one way to relate this concept is to reason about the difficulty of scheduling problems. We can say that when a scheduling problem satisfies certain property (i.e., when the system is in a certain phase), it is almost certain to schedule the task set and upon changing the property (the system enters a new phase), it is hardly possible to schedule the task set. For example, such a behavior has been observed for a uniprocessor non-preemptive scheduling problem. It has been shown that there exists a utilization threshold U^* such that, for large task sets, task sets with utilization $U < U^*$ can almost surely be scheduled and task sets with utilization $U > U^*$ almost surely cannot be scheduled [3]. It is also believed that such a behavior exists for identical multiprocessor scheduling problem [4]. We are interested in finding whether such a behavior exists for two-type heterogeneous multiprocessor platforms.

II. Open Problem

Does there exist a phase transition behavior for the two-type heterogeneous multiprocessor scheduling problem?

III. Some Insights

In a quest to find an answer to the question, we performed some simulations and did not observe the phase transition behavior in our simulations. We understand that these simulations/observations are not enough to answer the question and hence more work needs to be done in this regard. We brief our simulation setup and observations in this section.

We randomly generated the problem instances comprising the task set (with an upper bound of 15 tasks) and the computing platform (with an upper bound of 2 processors of each type). We then formulated the task assignment problem as Zero-One Integer Linear Program (ILP) as discussed in [1]. This formulation is shown in Figure 1. Here Z denotes the maximum capacity of any processor

Minimize Z subject to the following constraints :

C1.	$\sum_{j=1}^{m} x_i^j = 1$	$(i = 1, 2, \cdots, n)$
C2.	$\sum_{i=1}^{n} \left(x_i^j \cdot u_i^j \right) \leq Z$	$(j = 1, 2, \cdots, m)$
C3.	x_i^j is a non-negative **integer**	$(i = 1, 2, \cdots, n)$; $(j = 1, 2, \cdots, m)$

Figure 1. ILP formulation – ILP-Feas(τ, Π)

that is used and is set as the objective function (to be minimized). $Z \leq 1$ implies that the sum of utilization of tasks assigned to any processor is less than or equal to the available capacity on that processor – hence, $Z \leq 1$ indicates that the task set is feasible on the platform. The variable x_i^j (referred to as *indicator variable*) indicate the assignment of task τ_i to processor π_j, i.e., $x_i^j = 1$ implies that τ_i is (entirely) assigned to processor π_j, $x_i^j = 0$ implies that τ_i is not assigned to processor π_j. The first constraint ($C1$) indicates that every task must be assigned to processors. The second constraint ($C2$) indicates that no processor capacity should be used more than Z. The third constraint ($C3$) indicates that the indicator variables must be non-negative integers.

We extracted only those problem instances that are feasible (i.e., a problem instance in which the task set

could be assigned on the platform without missing any deadlines when EDF [2] is used to schedule the tasks on each processor – ILP solver returns $Z \leq 1$ for such task sets). For each of the feasible problem instances, we computed the *success ratio* which is defined as follows:

$$\text{success ratio} = \frac{N_{succ}}{N_{valid}}$$

where, N_{succ} denotes the number of assignments that meet all the deadlines of the tasks and N_{valid} denotes the total number of possible *valid* assignments. A valid assignment is one in which (i) no task is left unassigned and (ii) the task assignment in one valid assignment is different from other valid assignments. All the possible valid assignments are generated using exhaustive enumeration.

We then plotted our observations for 10000 feasible task sets with Z on X-axis and 'average success ratio' (for each Z) on Y-axis as shown in Figure 2. As we can see from the graph, there is a gradual decrease in the value of 'average success ratio' and hence no sharp threshold on a particular value of Z where 'average success ratio' reduces significantly. The fluctuation in the 'average success ratio' in the initial half of the graph (where $0 < Z \leq 0.45$) can be attributed to the fact that our task set generator generated very few task sets for which ILP solver gave the output $0 < Z \leq 0.45$ — to be precise, among 10000 task sets, only 250 were in this category. Hence, we believe that with a more balanced task set generator, we will not observe those fluctuations.

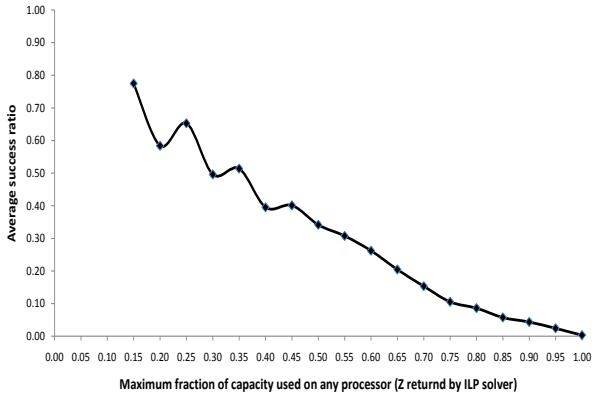


Figure 2. Average success ratio of randomly generated 10000 (feasible) task sets

From the observations made, we tend to believe that with the parameters that we have chosen, it may not be possible to observe the phase transition behavior if there is one. However, these are initial observations and we need to carry out more work to answer the following questions:

1) Does there exist a phase transition behavior? If there is one, then with what parameters we can observe such a behavior?
2) If there is no phase transition, then what is its implication considering the fact that such a behavior has been observed in the past for a uniprocessor scheduling problem [3][4]?

ACKNOWLEDGMENTS

This work was partially supported by the REHEAT project, ref. FCOMP-01-0124-FEDER-010045, funded by FEDER funds through COMPETE (POFC - Operational Programme Thematic Factors of Competitiveness), National Funds (PT) through FCT - Portuguese Foundation for Science and Technology and REJOIN project of FLAD (Luso-American Development Foundation).

REFERENCES

[1] S. Baruah, *Task partitioning upon heterogeneous multiprocessor platforms*, 10th IEEE International Real-Time and Embedded Technology and Applications Symposium (2004).

[2] C. L. Liu and J. W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*, Journal of the ACM (1973).

[3] S. Gopalakrishnan, M. Caccamo and L. Sha, *Sharp Thresholds for Scheduling Recurring Tasks with Distance Constraints*, IEEE Transactions on Computers (2008).

[4] S. Gopalakrishnan, *A sharp threshold for rate monotonic schedulability of real-time tasks*, 1st International Real-Time Scheduling Open Problems Seminar (2010).

Adaptive Schedulability Analysis

Luca Santinelli
INRIA Nancy Grand Est, Nancy, France
luca.santinelli@inria.fr

I. Introduction

Real-time systems are becoming highly dynamic hence with an explicit requirement of changing their behavior according to the needs encountered at run-time. *Actual real-time systems are required to adapt their functionalities depending on the external environment or their internal state.* So far it has been investigated the case when the applications are the system elements that can change their requirements, hence their functionalities, which fulfills just part of the requirements of complex and dynamic real-time systems. Indeed, whenever resource reservation (RR) mechanisms are applied to achieve temporal isolation among applications, the reservation parameters may need to change from one condition to another making the system even more complex. This implies that one of the main needs of actual dynamic real-time systems is the *adaptivity* of both the resource reservation and the applications. Consequently, the real-time analysis need a degree of flexibility to efficiently cope with such changing conditions.

The classical view of dynamic real-time systems refers to different operational modes designed to achieve different functionalities or to respond to changes of the system. Each mode specifies functional and non-functional characteristics and consists of specific resource requirements and resource availabilities. Thus, multi-moded real-time systems require a more accurate analysis than classical single-mode systems, because of the criticality of mode transitions. In fact, there are situations in which, although timing constraints can be guaranteed to be met within each individual mode (in steady state conditions), deadlines can still be missed during mode transitions. *The schedulability guarantees have to be provided during re-configurations, which in the multi-mode case are the mode transitions.*

A. Related Work

The problem of timing analysis across mode changes has been addressed in the real-time literature. First, mode-changing applications have been considered, examples are [1], [2], [3], with the underlying idea to wait for a certain amount of time δ before changing the schedule. The problem is then identifying a safe time instant where the new mode can be activated without causing deadline misses. However, it could be highly likely that all the changing applications require to change during the same idle time making such a extremely long so that it is impossible to guarantee specific timing behaviors. In such a case the analysis could be useless to solve probabilistic real-time problems.

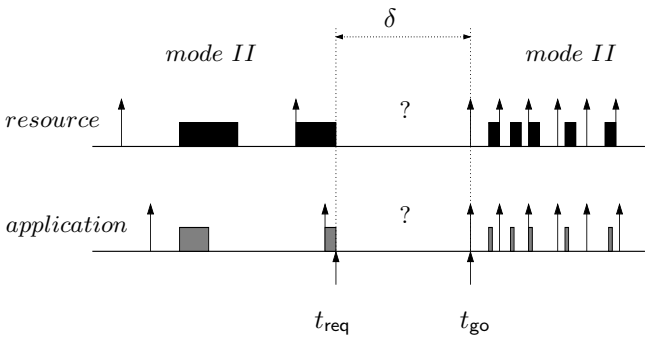


Figure 1. An example of transition with resource provisioning and application execution before and after the mode change t_{req}.

As we said, even resource reservation mechanisms can be affected by the mode-change. In that case, fixed reservation paradigms e.g. static reservations implemented with servers, [4], [5] are not appropriate to achieve the desired performance in case of applications with an highly dynamic resource demand. Recently, adaptive mechanisms have been proposed for servers. In [6], [7] respectively adaptive TDMA servers and periodic servers are studied from the mode-change perspective. The classical RR paradigms are changed to face dynamic conditions an derive the resource guarantees for the schedulability of the real-time applications. The server proposed are investigated during their mode change in order to derive the resource and to guarantee the schedulability of the real-time applications.

II. Schedulability Analysis

In order to investigate mode transitions in real-time systems it is necessary to characterize both the resource request of the applications and the resource provisioning of the RR mechanisms during such transitions from an old mode ($mode\ I$) to a new one ($mode\ II$). The multi-mode analysis assumes mode I and mode II feasible. Figure 1 describes a generic mode transition with t_{req} denoting the time instant at which the mode change is requested; the transition starts at t_{req}. From this time on, all the required changes in the system are initiated, while the new mode begins at t_{go} after a transition delay $\delta = t_{\mathsf{go}} - t_{\mathsf{req}}$. The transition finishes at t_{go}.

A possible abstraction for real-time systems refers to bounding functions in the interval domain. In literature it exists the following notions.

- The *workload bound function* $\mathsf{wbf}(t)$ for the task computational resource requirement (the workload) in the interval $[0, t]$. In case of fixed priority (FP) scheduling the workload bound function, in its level-i, $\mathsf{wbf}_i(t) = \sum_{j \in hp(i)} \left\lceil \frac{t}{T_i} \right\rceil C_i$ is the workload of the i-th task and all the high priority tasks than i, $hp(i)$.
- The *demand bound function* $\mathsf{dbf}(t)$, for the task minimum resource demand in order to meet its timing constraints (the resource demand). i.e. for a task τ_i it is $\mathsf{dbf}_i(t) = \max\left\{0, \left(\left\lfloor \frac{t-D_i}{T_i} + 1 \right\rfloor C_i\right)\right\}$. The computational demand of a task set Γ of periodic tasks synchronously activated at time $t = 0$ can be computed as the sum the individual demand bound functions of each task, that is $dbf_\Gamma(t) = \sum_{\tau_i \in \Gamma} dbf_{\tau_i}(t)$, expressing the total computation that must be executed by the processor in each interval of time when tasks are scheduled by earliest deadline first (EDF) paradigm. in order to meet all the task set constraints.
- The *supply bound function*, $\mathsf{sbf}(t)$ which is known as the minimum amount of resource provided (the resource provisioning).

With the bounding functions abstraction the schedulability criteria translates into bound comparison criteria, i.e. in case of EDF the schedulability of a task set within a resource provisioning is guaranteed if $\forall t\ \mathsf{dbf}(t) \leq \mathsf{sbf}(t)$, [8]. In case of FP, the schedulability is guaranteed if $\exists\ t_0 \in schedP_i$ / $\mathsf{wbf}_i(t_0) \leq \mathsf{sbf}(t_0)$; $schedP_i$ is the set of relevan points where to verify the schedulability, [9], [10].

With the bounding curves, there are the tuples $(\mathsf{wbf}^I, \mathsf{dbf}^I, \mathsf{sbf}^I)$ describing resource request and resource provisioning in mode I and $(\mathsf{wbf}^{II}, \mathsf{dbf}^{II}, \mathsf{sbf}^{II})$ for the resources in mode II. Besides, $(\mathsf{wbf}^T, \mathsf{dbf}^T, \mathsf{sbf}^T)$ describes the transition between modes in terms of workload, resource demand and resource provisioning. The schedulability has to be guaranteed in all the possible scenarios for the applications and the RR mechanisms, so the stable modes (mode I and mode II) as well as along the transitions.

The multi-mode analysis is one of the first results toward the schedulability for adaptive real-time systems. It relates on having different application demand bound functions and resource supply bound functions for each composing mode. As a cost, it demands to verify multiple scenarios and to compute multiple (and accurate) bounding functions. Furthermore, the analysis together with the mode abstraction, are limited to specific cases such as single transition per interval, [11].

III. Open Problems

Adaptive real-time systems demand a complete adaptive schedulability analysis which, among other requirements,

- *has to consider any possible system element changing in the system*;
- *it has to tackle with the case of complex transitions where many system elements could be asked to change at the same time*.

Furthermore, it could ask for alternative representations to the multi-mode one to explore further degrees of flexibility that adaptive systems have.

What else is required for having a mature enough adaptive schedulability analysis?

References

[1] K. W. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority pre-emptively scheduled systems," in *RTSS*, 1992, pp. 100–109.

[2] N. Stoimenov, S. Perathoner, and L. Thiele, "Reliable mode changes in real-time systems with fixed priority or edf scheduling," in *DATE*, 2009.

[3] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[4] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves for multimedia operating systems," Carnegie Mellon University, Pittsburg, Tech. Rep. CMU-CS-93-157, May 1993.

[5] X. Feng and A. Mok, "A model of hierarchical real-time virtual resources," in *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS 2002)*, December 2002, pp. 26–35.

[6] N. Stoimenov, L. Thiele, L. Santinelli, and G. Buttazzo, "Resource adaptations with servers for hard real-time systems," in *International Conference On Embedded Software (EMSOFT)*, 2010.

[7] L. Santinelli, G. Buttazzo, and E. Bini, "Multi-moded resource reservations," in *17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[8] S. Baruah, R. R. Howell, and L. E. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor," *Real-Time Systems*, vol. 2, 1990.

[9] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *IEEE Real-Time Systems Symposium*, 1989, pp. 166–171.

[10] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1462–1473, 2004.

[11] P. Pedro and A. Burns, "Schedulability analysis for mode changes in flexible real-time systems," in *ECRTS*, 1998, pp. 172–179.

Is Process Scheduling a Dead Subject?

Neil Audsley
Real-Time Systems Research Group, Dept. Computer Science, University of York, UK

The move towards multiprocessor commodity CPUs has led to a resurgence in multiprocessor scheduling research [1]. The main focus of this research has been the study of existing scheduling policies (eg. fixed priority, earliest deadline first, least laxity) when applied to typical small scale multiprocessor architectures (eg. multicore desktop machines). Promising advances have been made in scheduling policy and associated feasibility analysis. However, current research only addresses a constrained part of the more general multiprocessor scheduling problem. This short paper questions the fundamental assumptions behind current multiprocessor scheduling efforts and identifies other, more general areas of research required to match future system architectures and applications.

Assumptions

In general, current scheduling research (uniprocessor and multiprocessor) makes a number of fundamental implicit assumptions, including:

A1: The schedulable entity is the process (or task).

The process represents a unit of computation, for which some value of worst-case execution time (WCET) can be calculated or measured. The CPU (or CPUs) is multiplexed between the processes ensuring that all receive sufficient CPU time.

A2: The number of schedulable entities (ie. processes, tasks) is greater than the number of CPUs.

This assumption is the essence of the conventional scheduling problem - that CPUs are multiplexed between the application processes.

A3: The architecture is fixed i.e. the number and type of CPUs remains constant.

This assumption is partially relaxed when considering fault-tolerance in multiprocessor systems, where the feasibility of different numbers of CPUs is considered, noting that as CPUs are reduced there is a reduction in the amount of processes and/or the computation that the processes perform. This assumption is also considered by energy aware scheduling, where the performance of the CPU can be reduced whilst still enabling all processes to meet their deadlines.

A4: The memory hierarchy forms a single static uniform shared address space.

The WCET for a process must consider the memory hierarchy to determine load times, cache or scratchpad behaviour, and/or cache consistency (for typical multiprocessor architectures).

Architectural Trends

Whilst these assumptions have been adopted for conventional real-time systems research, they should be questioned in the light of current architectural trends and future applications. Architectural trends include:

T1: Massive parallelism.

The degree of potential parallelism available within a single chip is increasing almost exponentially. Current commodity CPUs, with conventional memory hierarchies, contain upto 16 cores. They represent a scaling of conventional SMP architecture with cache consistency. This approach is known to be limited in scalability due to the expense of cache-consistency logic. General purpose massively parallel architectures, such as Tilera (100 cores) [7] are based upon Network-on-Chip (NoC) [2] concepts. Within single chips upwards of around 1K CPUs can be achieved with todays fabrication technologies, with active research into larger (10K+ CPU) devices ongoing [3].

T2: Simpler CPUs.

As more CPUs are placed within a single chip, energy provision to, and cooling of, the CPU core become critical dictating that slower simpler cores will be used, with lower amounts of cached (or scratchpad) memory than seen in todays commodity packages.

T3: Non-uniform Memory Architectures (NUMA).

As the number of CPUs within a chip scales, there are practical limitations on the uniformity of the memory structure - in essence, physical memory locations will be at different distances (in terms of cycles) from each CPU. Such NUMA architectures are already seen in (some) desktop commodity CPU multicores (so-called ccNUMA architectures). Whilst NUMA architectures are often presented to the programmer as a uniform address space (eg. in conventional multicore desktop systems), NUMA (and more extreme architectures) are essentially message passing between islands of coherent memory space [5], posing challenges to WCET and schedulability analyses.

T4: Heterogeneous architectures.

In general, the CPUs within multicores will have different capabilities. This could be merely the presence or not of an floating point unit on some CPUs, but is more likely to include specialisations of CPUs for specific purposes, eg. the inclusion of a SIMD unit to aid vector processing. Different CPUs entirely can be included within the same chip, eg conventional, GPU, DSP. Function accelerators

can also be included (eg. for I/O processing, or communications). The architecture can also be heterogeneous in terms of the communications - particularly for NoCs. Examples of heterogeneous architectures today include Cell (different CPUs and communications); various mixed CPU/GPU architectures (eg. ARM Tegra).

T5: Reconfigurable architectures.

Increasing flexibility is being built into single and multiprocessor architectures. Driven by embedded systems and the need for specialist functionality, Application Specific Instruction Processors (ASIPs) [4] are increasingly available, with some portion of the instruction set defined by the application. Also, CPUs and FPGA technology on the same chip [6] enable application specific function accelerators to be defined. Pure FPGAs represent the extreme reconfigurable architecture, and can be reconfigured dynamically at run-time to be different CPUs, or mixtures of CPUs and function accelerators.

CHALLENGES

The architectural trends summarised above challenge the fundamental assumptions of scheduling theory. Two main challenges are now articulated:

C1: Schedule data not threads.

Massive parallelism (T1) will lead to a number of CPUs greater than the number of application processes, breaking (A2). Even if some of the additional parallelism is used for achieving fine-grained parallelism within a process (ie. parallel for loop), the ratio of CPUs to processes is increasing, to at least 1.

When it is no longer necessary to schedule processes on CPUs (breaking (A1)), the fundamental challenge is to schedule the movement of data and code around the architecture. The trend towards NUMA (T3) makes this a difficult problem, as memory structures and hierarchies are no longer uniform, breaking (A4). Additionally, simpler CPUs (T2) imply less local memory adjacent to a particular CPU, removing the assumption often made (particularly in NoC research) that there is sufficient local memory to hold all code. Hence the challenge is to schedule the movement of data (including code).

This challenge is also supported by application trends towards faster larger data I/O. This is exemplified by handheld mobile devices, whose data-rates are increasing exponentially from low quality sound / data through to high-definition. The movement of this data into the platform, processing the data, and outputting data, has become the dominant design consideration.

C2: Time-Space and Heterogeneous Scheduling.

Heterogeneous (T4) and reconfigurable architectures (T5) break assumptions over architectures being fixed (A3). They enable the fundamental computer science trade-off of time-space to be examined within the context of real-time system scheduling. Essentially, it allows for different implementations of an application, using different amounts of hardware capacity - in general, the more hardware resource used, the shorter the execution time. When these decisions can be made dynamically at run-time (noting reconfiguration overheads), then the scheduling problem is widened to including the architecture itself.

This challenge suggests a movement towards joining scheduling research with that of hardware / software co-design research (where application specific architectures are defined) and with that of embedded systems (where systems are designed with limited space and energy constraints).

BIBLIOGRAPHY

[1] R.I. Davis and A. Burns, *"A Survey of Hard Real-Time Scheduling for Multiprocesor Systems"* , ACM Computing Surveys, 2010

[2] C. Nicopoulos, V. Narayanan and C.R. Das, *Network-on-Chip Architectures*, pub Springer LNEE, vol 45, 2010

[3] *"Teraflux: Exploiting Dataflow Parallelism in Teradevice Computing"*, EU FP7 Project, http://www.teraflux.org, 2011

[4] T. Gloker and H. Meyr, *Design of Energy-Efficient Application Specific Instruction Set Processors (ASIPs)*, Kluwer, 2004

[5] I. Gray and N. Audsley, *"Targeting Complex Embedded Architectures by Combining the Multicore Communications API (MCAPI) with Compile-Time Virtualisation"*, ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES), 2011

[6] Xilinx Zynq-7000 FPGA, http://www.xilinx.com/technology/roadmap/zynq7000/features.htm, 2011

[7] Tilera Multicore Processor, http://www.tilera.com, 2011

Using Schedulability Analysis Techniques to Derive the Average-Case Behavior with a Monte-Carlo Simulation

Steffen Kollmann, Victor Pollex and Frank Slomka
Institute of Embedded Systems/Real-Time Systems
Ulm University
{firstname.lastname}@uni-ulm.de

I. Motivation

Evaluating the time-behavior of distributed embedded real-time systems can be performed in many ways. Mainly three different approaches can be used as shown in figure 1. The first one is to test the implemented system, the second one is to perform a simulation and the third one is to conduct a real-time analysis. Each of them has its advantages and disadvantages.

Testing the implemented system is certainly the most realistic approach concerning the results, because the final system is used. But correcting any failures discovered at this late design stage can lead to enormous costs. To prevent this situation other approaches have to be used that are suitable for early design stages where mistakes are less expensive to fix.

Such methods are the simulation or the analysis of a system. These can also be used in late design phases. For example a tool like chronSIM [1], which allows simulating a system, can provide information on what a system is actually doing especially in the average case. With a simulation however the border cases like the best-case and worst-case time behavior cannot be determined. Due to the coverage problem that simulations have, it is unknown whether such a border case has been simulated or not. Analytical approaches like those based on Tindell and Clark [2] construct the border cases and calculate guaranteed bounds for the time behavior. In [3] an automotive case study was conducted where both approaches - simulation and analysis - are compared. The conclusion of the paper is that both approaches are needed for the design process

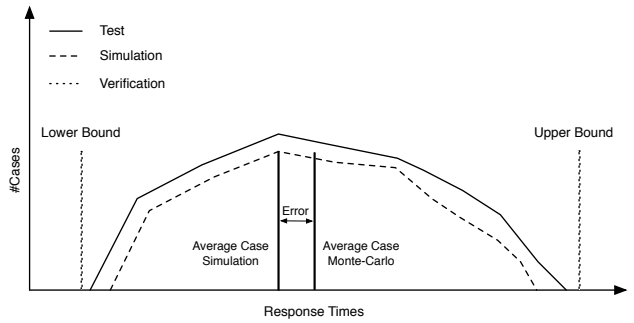


Figure 1. Different evaluation techniques

of embedded hard real-time systems. The focus in the paper is to determine the worst-case time behavior. But to consider only the border cases during a system design is often not sufficient to construct a reliable system.

An example for this can be found in the domain of control systems. To design a reliable controller on the one hand the border cases are required to verify the stability of the control loop. On the other hand the most important part in the design is the knowledge of the average-case time behavior, because a controller should perform best in the average case. Therefore both values are needed to design a reliable controller. In [4] the impact of the time behavior on a control system is discussed more in detail.

As mentioned with a simulation the average case can be obtained, because it will most likely describe the actual system behavior. However the more complex a system is, the longer will be the runtime of the simulation. A good impression about the costs required to perform a simulation can be obtained by considering the results in [3]. The runtime can easily be in the magnitude of days to get significant data regarding the average case due to it depending on the history of the simulation. Therefore such an approach cannot be parallelized easily. It is possible to start various simulation runs simultaneously, but it is not clear how long each run has to be executed in order to obtain the desired data. So the question arises whether other methods can be used to determine the average case while needing less time than a simulation.

Analytical approaches currently only construct the border cases and calculate guaranteed bounds for them. Naturally the average case must be between the calculated best case and worst case. The question is if an analytical approach like the SymTA/S approach [5] or the real-time calculus [6] can be modified or extended in such a way that the average case or at least an approximation of the average case can be obtained more quickly than with a simulation. In [7] an extension of the real-time calculus is proposed where probabilistic arrival and service curves are considered. But it is an open question if the approach is able to construct the average-case behavior, because with the presented model it seems improbable that the average case of the system can be found.

We now sketch a proposal on how an analytical ap-

proach can be modified to possibly obtain the average case. The idea is based on a Monte-Carlo simulation [8] and to the best of our knowledge there is no successful work which describes how schedulability analysis techniques can be used with a Monte-Carlo simulation to compute the average-case behavior of a system.

In [9] a technique is presented which is based on a similar idea. A Monte-Carlo simulation is used to determine the time-behavior of a system, but it is not clear which method is used for the Monte-Carlo simulation. It is stated that the network-calculus is not used, because the system behavior cannot be modelled adequately.

II. Proposal

We now present our proposal and start our discussion with the simulation approach, because it is reasonable to understand why it is able to find the average case. The advantage of such a method is that the different correlations/dependencies between the tasks are covered inherently. But dependencies within a system are an issue that analytical approaches have. Ignoring them in the analysis results in more pessimistic bounds, therefore as many dependencies as possible must be considered to obtain an average case. Many approaches have been developed to consider data dependencies [6] or task dependencies [10]. It is however unclear whether the existing work regarding the dependencies is sufficient to be able to calculate the average-case behavior.

The question is now how the average case can be determined efficiently. We propose to use a Monte-Carlo simulation based on a schedulability analysis by varying parameters like stimulation, execution time, etc. The stimulation may only be varied within its bound as well as the execution time may only be varied between its best-case and worst-case execution time. To be able to freely vary the stimulation the periodic model with jitter is insufficient, instead more expressive models have to be used like the event streams [11] or the arrival curves used by the real-time calculus [6]. Some questions which have to be answered to obtain the average-case behavior are:

- Is a Monte-Carlo simulation based on a schedulability analysis even suitable to obtain the average-case behavior?
- How must the parameters be randomized?
- Which dependencies must be modeled in order to obtain acceptable results?
- How many samples have to be gathered in order to be able to approximate the average case?
- Is it possible to bound the error of the approximation?

The first question addresses the fact that analytical approaches always construct the corner cases. Is it therefore possible to obtain the average case with a Monte-Carlo simulation? The next question considers the issue of how the parameters for the Monte-Carlo simulation have to be randomized. Can we use the same method as used by simulation approaches or do we have to consider anything else? To approximate the behavior of a simulation, task dependencies have to be considered during an analysis. So which are the important dependencies that influence the time behavior? The last two questions relate to the results of the Monte-Carlo simulation, where the quality of the results has to be determined and a threshold has to be set for which the quality is deemed acceptable.

References

[1] chronSIM, http://www.inchron.com.

[2] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, pp. 117–134, April 1994.

[3] S. Kollmann, V. Pollex, K. Kempf, F. Slomka, M. Traub, T. Bone, and J. Becker, "Comparative application of real-time verification methods to an automotive architecture," in *Proceedings of the 18th International Conference on Real-Time and Network Systems*, 2010.

[4] T. Bund, S. Moser, S. Kollmann, and F. Slomka, "Guaranteed bounds for the control performance evaluation in distributed system architectures," in *Proceedings of the International Conference on Real-Time and Embedded Systems (RTES 2010)*, Singapore, Sep. 2010.

[5] K. Richter, "Compositional scheduling analysis using standard event models - the symta/s approach," Ph.D. dissertation, University of Braunschweig, 2005.

[6] E. Wandeler, "Modular performance analysis and interface-based design for embedded real-time systems," Ph.D. dissertation, ETH Zurich, September 2006.

[7] L. Santinelli and L. Cucu-Grosjean, "Toward probabilistic real-time calculus," *SIGBED Rev.*, vol. 8, pp. 54–61, March 2011.

[8] D. Kroese, T. Taimre, and Z. Botev, *Handbook of Monte Carlo Methods*. Wiley, 2011, vol. 706.

[9] C. Mauclair, "Analysis of Real-Time Networks with Monte Carlo Methods," http://sites.onera.fr/journeesdestheses-tis/sites/sites.onera.fr.journeesdestheses-tis/files/articles/JDT-TIS-2011_Mauclair_Cedric.pdf, 2011.

[10] J. C. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *RTSS*, 1998, p. 26 ff.

[11] K. Gresser, "An event model for deadline verification of hard real-time systems," in *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, 1993, pp. 118–123.