RTSOPS 2010

Proceedings of the 1st International Real-Time Scheduling Open Problems Seminar

Brussels, Belgium
July 6, 2010

In conjunction with:
The 22nd Euromicro Conference on Real-Time Systems (ECRTS10),
July 6-9, 2010

Edited by Robert I. Davis and Nathan Fisher

Foreword

Welcome to Brussels and the 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS 2010). This new 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 2010 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. These proceedings are also published as a Technical Report from the University of York, Department of Computer Science (YCS-2010-455) available at http://www.cs.york.ac.uk/ftpdir/reports/2010/YCS/455/YCS-2010-455.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	Polytechnic Institue of Porto (Portugal)
Sanjoy Baruah	The University of North Carolina at Chapel Hill (USA)
Marko Bertogna	Scuola Superiore Sant'Anna, Pisa (Italy)
Liliana Cucu-Grosjean	INRIA Nancy-Grand Est (France)

Special thanks also go to Jim Anderson, Joël Goossens, Vandy Berten, and Gerhard Fohler for their support and assistance in organising this seminar.

Robert Davis and Nathan Fisher
Co-chairs
1st International Real-Time Scheduling Open Problems Seminar (RTSOPS 2010)

Table of Contents

Conjecture about global fixed-priority preemptive multiprocessor scheduling of implicit-deadline sporadic tasks: The utilization bound of SM-US($\sqrt{2}-1$) is $\sqrt{2}-1$

Björn Andersson
CISTER/IPP-Hurray Research Unit at the Polytechnic Institute of Porto
Email: bandersson@dei.isep.ipp.pt

***Abstract*—Consider global fixed-priority preemptive multiprocessor scheduling of implicit-deadline sporadic tasks. I conjecture that the utilization bound of SM-US($\sqrt{2}-1$) is $\sqrt{2}-1$.**

I. Preliminaries

Consider the problem of preemptively scheduling n sporadically arriving tasks on $m \geq 2$ identical processors. A task τ_i is uniquely indexed in the range 1..n and a processor likewise in the range 1..m. A task τ_i generates a (potentially infinite) sequence of jobs. The arrival times of these jobs cannot be controlled by the scheduling algorithm and are a priori unknown. We assume that the arrival time between two successive jobs by the same task τ_i is at least T_i. Every job by τ_i requires at most C_i time units of execution over the next T_i time units after its arrival. We assume that T_i and C_i are real numbers and $0 \leq C_i \leq T_i$. A processor executes at most one job at a time and a job is not permitted to execute on multiple processors simultaneously. The utilization is defined as $U_s = (1/m) \cdot \sum_{i=1}^{n} \frac{C_i}{T_i}$. The utilization bound UB_A of an algorithm A is the maximum number such that all tasks meet their deadlines when scheduled by A, if $U_s \leq UB_A$.

Global fixed-priority preemptive scheduling is a specific class of algorithms where each task is assigned a priority, a number which remains unchanged during the operation of the system. At every moment, the m highest-priority tasks are selected for execution among tasks that are ready to execute and has remaining execution. The scheduling decisions are therefore determined by the assignment of priorities to tasks. The priority-assignment scheme in the current state-of-art which offers the highest utilization bound is SM-US($2/(3+\sqrt{5})$; its utilization bound is $2/(3+\sqrt{5})$ [3]. It categorize a task as heavy or light. A task is said to be heavy if $\frac{C_i}{T_i}$ exceeds $2/(3+\sqrt{5})$ and a task is said to be light otherwise. Heavy tasks are assigned the highest priority and the light tasks are assigned a lower priority; the relative priority order among light tasks is given by SM; slack monotonic, meaning that if $T_i - C_i < T_j - C_j$ then τ_i is given higher priority than τ_j.

One can show (last page of [1]) that each priority assignment scheme which is scale-invariant and independent has a utilization bound at most $\sqrt{2}-1$. (A priority-assignment scheme is scale-invariant if the relative priority order of a priority assignment given does not change when we multiply T_i and C_i of all tasks by the same positive constant. A priority-assignment scheme is independent if $\text{priority}_\text{i} = f(T_i, C_i)$, that is the priority of a task τ_i depends only on its own parameters.)

II. The conjecture

Let SM-US($\sqrt{2}-1$) denote a priority-assignment scheme which categorized a task as heavy if $\frac{C_i}{T_i}$ exceeds $\sqrt{2}-1$ and a task is categorized as light otherwise. Heavy tasks are assigned the highest priority and the light tasks are assigned a lower priority; the relative priority order among light tasks is given by SM; slack monotonic, meaning that if $T_i - C_i < T_j - C_j$ then τ_i is given higher priority than τ_j.

I conjecture that the priority-assignment scheme SM-US($\sqrt{2}-1$) has the utilization bound $\sqrt{2}-1$.

III. Significance of the conjecture

If the conjecture would be true then we would have at our disposal a priority-assignment scheme that attains the best performance possible in the class of scale-invariant and independent priority-assignment schemes.

IV. The rationale for stating the conjecture

We can understand this conjecture by considering two task set examples. As a first example, consider tasks τ_1, τ_2, ..., τ_m with $T_i = 1$,$C_i = \sqrt{2}$-1 and $T_{m+1} = \sqrt{2}$ and $C_{m+1} = 2-\sqrt{2}$ to be scheduled on m processors. For these tasks, it holds that the utilization of each task is $\sqrt{2}-1$. Increasing the execution time by an arbitrarily small amount will cause a deadline miss for the case that all tasks arrive simultaneously. We conclude from this example that we cannot prove a higher utilization bound than $\sqrt{2}-1$ for the algorithm SM-US($\sqrt{2}-1$).

References

[1] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%", Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03), pp. 33 - 40, 2003.

[2] L. Lundberg, "Analyzing Fixed-Priority Global Multiprocessor Scheduling", Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), pp. 145-153, 2002.

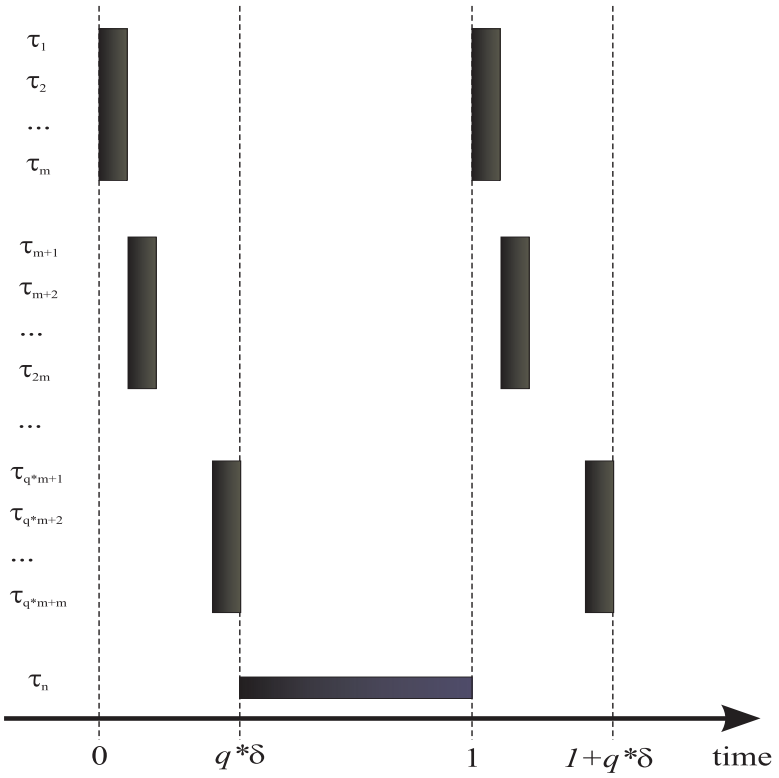


Fig. 1. [Adapted from [2]] An example of a task set where RM-US(0.375) performs poorly. All tasks arrive at time 0. Tasks τ_1, τ_2,. . ., τ_m are assigned the highest priority and execute on the m processors during [0,δ). Then the tasks τ_{m+1}, τ_{m+2},. . ., τ_{2m} execute on the m processors during [δ,2δ). The other groups of tasks execute in analogous manner. Task τ_n executes then until time 1. Then the groups of tasks arrive again. The task set meets its deadlines but an arbitrarily small increase in execution times causes a deadline miss.

[3] B. Andersson, "Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%", Proceedings of the 12th International Conference On Principles Of DIstributed Systems (OPODIS'08), pp. 73-88, 2008.

Dual Priority Scheduling: Is the Processor Utilisation bound 100%?

Alan Burns
Department of Computer Science
University of York, UK

EXTENDED ABSTRACT

Fixed priority (FP) schemes have the disadvantage that processor utilisations less than 100% must be tolerated if a system is to be guaranteed off-line. By comparison earliest deadline first (EDF) scheduling can theoretically utilise all of a processor's capacity. In this paper the dual priority scheme is revisited: here a task may execute in two phases; each phase has a static priority assigned, the transition from one phase to another is made at a fixed offset in time from the release of the task.

In their seminal paper of 1973 Liu and Layland [2] introduced the fixed priority (FP) scheme known as *rate monotonic priority assignment*, and the dynamic priority scheme known as *earliest deadline first* (EDF). Their theoretical treatment produced the well known utilisation (U) bounds for a set of n periodic tasks with period T equal to deadline D; and computation time, C. For FP the bound converges, approximately, on the value 0.69 (69%) for large n. For n equal to 2 it is close to 0.83. For EDF the bound is 1 for all n.

In 1993 a dual priority assignment scheme was introduced [1]. This paper contained the conjecture:

Conjecture 1: For any task set with total utilisation less than or equal to 100% there exists a dual priority assignment that will meet all deadlines.

This remains an open question. Search techniques have failed to find a counter example. Even if one exists then there remains the question as to what is the utilisation bound? is this is a function of the number of tasks (n)? and if not 'dual' priority then 'triple'?, or 'quad'?, or what? At some level (m), 'm-priority' assignment can be made to emulate EDF and hence there does exist a 'static' scheme that can obtain the 100% bound – but is it the dual priority scheme?

The dual priority scheme is a minimal dynamic scheme that allows a task to change (increase) its priority during its execution. Each task has at most two priority levels: many tasks will continue to have only one priority. At run-time only standard preemptive priority based scheduling is required. A standard RTOS (with a priority change primitive – which is a commonly supported feature) could therefore support task sets with the same utilisation bound as EDF. A dual priority scheme could also be used with priority-based non-preemptive communication protocols such as CAN. Here an EDF-based protocol is not possible, but a dual-priority scheme, in which the priority of a message is increased if it has been in a node's output buffer for a predefined interval of time, is certainly feasible with only a minor change to the CAN protocol.

In addition to the usual notation each task has an intermediate deadline S_i at which time it undergoes a step change (increase) in priority. For all tasks: $0 \leq S_i \leq T_i$. Each task has a *phase 1* priority P_i^1 and if $S_i < T_i$ a *phase 2* priority P_i^2 with $P_i^1 < P_i^2$.

The following assumptions are used. Any task, in either phase, can be preempted by any other task running at a higher priority. Tasks do not suspend themselves other than at the end of their computations. The time required to perform context switching, priority changes etc is ignored (i.e. assumed to be zero). A single processor is assumed.

A simple example will illustrate the benefits of this scheme. Table I gives the details of a three task system, each task has T = D. Note that priority 1 is high and 4 is low. The total utilisation is 100%, and the LCM of the task periods is 24.

	T	C	P	P^2	S	U
τ_1	6	3	2			50%
τ_2	8	2	3			25%
τ_3	12	3	4	1	11	25%

TABLE I
EXAMPLE TASK SET

If the dual phasing is ignored (i.e. the tasks are treated as having single priorities) then the task set is not schedulable by rate monotonic priority assignment (or any other static priority assignment scheme as rate monotonic is optimal). The lowest priority task (τ_3) can only execute for two ticks before its first deadline. But if τ_3 has its priority raised at tick 11 to above τ_2 then all deadlines are met.

Another task set that requires all tasks to have a priority change is given in Table II.

	T	C	P	P^2	S	U
τ_1	28	21	4	1	9	75%
τ_2	100	15	5	2	84	15%
τ_3	160	16	6	3	130	10%

TABLE II
EXAMPLE TASK SET

The current state of the dual priority conjecture is:

- For $n = 2$ a proof has been obtained (so $U = 1$ rather than $U = 0.83$ for standard FP) – see Appendix.

- No counter example found with extensive searches for $n = 3$.
- No counter example found for $n > 3$, but search is computationally expensive as:
  - Simulation up to the LCM needed as the scheduling test.
  - No formulae exists for computing the migration points (the Ss).
- The phase 1 priorities are probably Rate Monotonic.
- The phase 2 priorities are possible also Rate Monotonic with all phase 2 priorities higher than all phase 1 priorities.
- For some task sets the highest priority tasks may have $S_i = 0$.
- A range of promotion points may all lead to a schedulable system.
- The promotion points are a function of task computation times (ie. not just task periods).

The open question is therefore: is the utilisation bound for the dual priority scheme 1? And if it is, how are the promotion points (Ss) computed? If it is not, what is the bound for the dual priority scheme, and is there a m-priority scheme that does provide the maximum bound – and is m then a function of n?

One possible method of tackling this question is to consider the behaviour of the EDF scheme. Whilst EDF tasks do not have static priorities, EDF jobs do. There is therefore a fixed number of partial orders for job executions. Does the dual priority scheme have a similar number?

REFERENCES

[1] A. Burns and A. J. Wellings. Dual priority assignment: A practical method of increasing processor utilisation. In *Proceedings of the Fifth Euromicro Workshop on Real-Time Systems, IEEE Computer Society Press*, pages 48–53, Oulu, Finland, 1993.

[2] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.

APPENDIX - PROOF FOR TWO TASKS

Consider two tasks, τ_1 and τ_2 with $T_1 < T_2$ and hence priority of τ_1 greater then the priority of τ_2. Assume total utilisation is the worst-case value of 1:

$$C_1/T_1 \ + \ C_2/T_2 \ = \ 1 \tag{1}$$

Assume τ_2 has a promotion point at time g before its deadline (ie. $g = T_2 - S_2$). For a two task system τ_1 does not need to be promoted. Consider an arbitrary deadline of τ_2, as $D_2 = T_2$ then this deadline can be represented by the time pT_2 – for some value of p (assuming the system starts its execution at time 0). Finally let l be the shortest interval from a previous release of τ_1 and the point pT_2 – see figure 1.

At point pT_2, τ_2 should should have executed for time pC_2; τ_1 will have executed for a number of complete invocations:

$$\left((pT_2 - l)/T_1\right) C_1$$

plus a partial computation of maximum size: $l - g$. This implies that:

$$\left((pT_2 - l)/T_1\right) C_1 + pC_2 + l - g \ \leq \ pT_2 \tag{2}$$

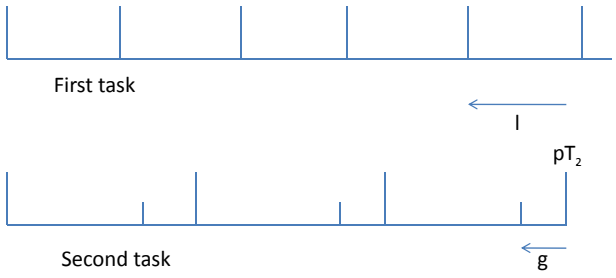


Fig. 1. Two task execution

Substituting for C_1 from Eqn (1) in Eqn (2) produces:

$$(pT_2 - l)(1 - C_2/T_2) + pC_2 + l - g \ \leq \ pT_2$$

Expanding the first terms and canceling a number of balanced terms allow this relation to be simplified to:

$$lC_2/T_2 \leq g$$

Now the value l depends on which deadline of τ_2 is considered, but its maximum value is the maximum distance between a multiple of T_1 a corresponding 'next' release of τ_2. Hence the maximum value of l is equal to $T_1 - H$, where H is the highest common factor of the integers T_1 and T_2. This gives the final bound on g of

$$(T_1 - H)C_2/T_2 \ \leq \ g \tag{3}$$

Note if T_2 is a multiple of T_1 then H is equal to T_1 and g can be zero, ie. no promotion point needed. This result would be expected as task sets with utilisation of 1 are schedulable in this circumstance. Also if T_1 and T_2 are co-primes then Eqn (3) becomes:

$$(T_1 - 1)C_2/T_2 \ \leq \ g$$

The other constraint on g is that τ_1 must remain schedulable at its deadlines. Here the worst case is when two deadlines coincide. Now all of time g can be used by τ_2, and hence

$$g \ \leq \ T_1 - C_1 \tag{4}$$

Now a value of g must exist (and hence the two tasks be schedulable) if Eqns (3) and (4) can both be satisfied for any feasible task set (ie. one satisfying Eqn(1)). If this were *not* the case then

$$T_1 - C_1 \ < \ (T_1 - H)C_2/T_2,$$

which implies

$$T_1 - C_1 \ < \ (T_1)C_2/T_2,$$

giving

$$1 - C_1/T_1 \ < \ C_2/T_2,$$

and hence

$$1 \ < \ C_2/T_2 + C_1/T_1$$

which is clearly false.

This provides the proof that all two task systems have a dual priority scheme that will guarantee schedulability if the utilisation of the tasks is feasible (not more than 1). Moreover, the promotion time is provided by any value of g satisfying Eqns (3) and (4).

As an example consider a simple task set with $T_1 = 8$, $C_1 = 4$, $T_2 = 12$ and $C_2 = 6$; H has the value 4 and so the promotion point for τ_2 must be at least 2 (from its deadline) and no more that 4. A simple simulation shows that values of 2, 3 and 4 will all lead to schedulability.

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

Sebastian Altmeyer, Claire Maiza
Saarland University, Germany
{altmeyer,maiza}@cs.uni-saarland.de

In real-time systems, tasks must obey stringent timing constraints. A verification process that checks if these constraints are met consists of a timing analysis of each task and schedulabiltiy analysis of the set of tasks. The interface between these two analyses is the task model constituting an abstraction of the task's timing properties in the system. A very basic task model was presented by Liu and Layland [4]: the execution demand of a task i, often denoted as C_i, abstracts all possible execution times to a single value. The aim of the timing analysis is to compute this abstraction of the timing behavior of the tasks by safely bounding their worst-case execution time (C_i).

By the abstraction step from timing analysis to task model some precision is lost. Especially in preemptive systems or systems with interrupts, timing analysis computes, in addition to the pure time bound for uninterrupted execution, the additional delay due to interrupts or preemptions. As research on this topic has shown [1], preemption 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, 6], or for the n^{th} preemption of task i, or for preemption occurring at point p [1, 3]. If a schedulability analysis is able to take into account such precise information about the preemption costs, the results may exhibit a higher precision.

However, schedulability analyses are often based either on the basic task model by Liu and Layland with a unified bound on the execution time including preemption costs or on a model using only one separated value for the preemption costs per task [5]. The second task model improves over Liu and Layland's model by distinguishing preemption costs depending on the actual number of preemptions instead of considering an upper bound. Nevertheless, both models exhibit an inherent pessimism. The bound on the additional preemption delay—no matter if part of the execution time bound or considered separately—must comprise all possible preemption scenarios regarding preemptions points, preempting task etc., even if they do not occur in the actual schedule. However, schedulability analysis generally uses a simple model like the one of Liu and Layland to reduce the complexity of the schedulability test.

So, on the one hand, the abstraction of the timing behavior in the task model comes at the cost of inherent pessimism and on the other hand, schedulability analysis may rely on a simplified task model to reduce complexity. The tradeoff between precision and complexity of the schedulability analysis is determined by the task model and its abstraction of the timing of tasks. This tradeoff raises some questions:

- How high is the inherent pessimism and imprecision of a specific task model due to the precision of the abstraction of the timing properties?
- What is a good tradeoff between precision of the task model and complexity of the schedulability analysis?
- How to integrate such detailed information about the timing of tasks in the schedulability analysis?

Furthermore, incorporating more precise information, such as preemption points or preempting task, may be infeasible in general; thus, the following question arises in this context:

- Is it possible to adapt the schedule or the system in order to better benefit from the precision provided by timing analysis?

A typical example in which the system is adapted to achieve higher precision is the use of deferred preemption, i.e., preemption limited to predefined program points. Although flexibility of the schedule is lost to some degree, schedulability may be achieved only due to a strongly reduced bound on the preemption costs.

Answers to the questions listed above enable a better understanding of the influence of preemptions costs on the schedule and provide guidelines for the design and schedulability analysis of real-time systems. Note that the tradeoff between precision of the task model and complexity of the schedulability analysis becomes especially apparent in case of preemptive systems—but is not limited to such. Other issues where more precision from timing analysis could be taken into account are for instance the cache contention for multicore systems or the difference between first and all further executions of one task, which often strongly varies due different initial cache states.

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] 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*.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1), 1973.

[5] P. Meumeu Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *ECRTS 2007*.

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

On the Complexity of PROFINET IRT Scheduling

Olaf Graeser
Institut Industrial IT
Liebigstrasse 87
32657 Lemgo
Germany
Email: olaf.graeser@hs-owl.de

Oliver Niggemann
Institut Industrial IT
Liebigstrasse 87
32657 Lemgo
Germany
Email: oliver.niggemann@hs-owl.de

***Abstract*—In industrial automation, fieldbus systems are used to enable decentralised factory automation and process control. These fieldbusses are specialised communication systems tailored to meet the requirements of these applications. Such requirements are often hard real–time constraints which can only be guaranteed using deterministic communication systems like time–triggered networks. Such deterministic communication systems require a preplanning of the communication timing during the design phase of the automation system.**

In our work, the communication system Profinet IRT will be formalised as an example of a time–triggered network. Furthermore the complexity of the communication planning (scheduling) will be investigated with respect to different network topologies and different variants of the scheduling problem.

I. Introduction

Current communication systems in industrial automation are often based on Ethernet. Thereby, the communication technology on the factory floor (field level) is the same as in the office network of a company. This allows to have compatible communication technologies across all levels of automation and supports therefore a seamless vertical communication between all these levels. Hence,at least in theory, even an office PC can collect manufacturing process data from a field device on the factory floor.

Examples of such communication systems for the field level are PROFINET [Pop05], EtherCAT [JB03], SERCOS III [SER08] and Ethernet/IP [Ope10]. Since the Ethernet standard was not developed with respect to the requirements of industrial automation, the mentioned fieldbus systems must face some technical challanges [Jan02], [Fel00]. However, by modifications in the Ethernet's MAC layer it is possible to comply with the typical hard real–time requirements of factory automation [Dop08], [Jas05].

II. Open Problem

In this work, PROFINET IRT (Isochronous Real–Time) will be used as an application example of modified switched Ethernet which complies with hard real–time requirements but, at the same time is compatible with standard Ethernet. This is achieved by using two communication phases: One phase for real–time communication and another phase for non real–time communication (see Fig. 1). To guarantee deterministic transmission times, during the the real–time phase a Time–Division–Multiple–Access (TDMA) approach is used.

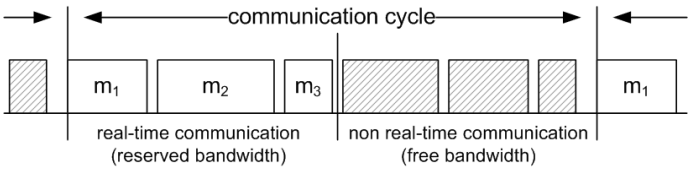


Fig. 1. Communication phases in Profinet IRT

By minimising the length of the real–time communication phase, more bandwidth becomes available for the other phase or for additional real–time communication. This minimisation task requires a preplanning of communication throughout the network.

For each message, the route, the release time, and the start times for forwarding this message in every single switch on the planned route must be preplanned. Since "cut through" switching is more performant and therefore preferable to "store and forward" switching, the start times for forwarding depend on the release time of the message and the transmission delays on the route.

An additional possibility to reduce the length of the real–time communication phase is to use macro cycles. The idea is, that a message might not be sent in every cycle, but only every n–th cycle.

The complexity of this complete preplanning depends on the traffic in the network, the usage of macro cycles and, particularly, on the network topology.

III. State of the art of Profinet IRT Scheduling

The work in [Dop08] describes an approach, starting with a simplified, PROFINET IRT communication system that uses conflict graphs in combination with edge and node colouring to derive local communication schedules for each network switch. These local schedules then have to be synchronised. This approach with local schedules does not work for a real PROFINET IRT network with variable frame length, broadcast– and multicast messages, because in this case a global conflict graph is necessary and the exact colouring of such a graph is NP-complete.

The work in [HBv09] formulates the PROFINET IRT scheduling problem as Resource Constrained Project Scheduling with Temporal Constraints. Messages are represented as

chains of tasks where each task stands for the communication over an link between network switches. Each message has a release date, an end–to–end delay and a deadline. In the first step, these constraints are transferred into a directed and weighted (constraint–) graph. In the second step a solver finds a suitable schedule for the constraint graph. The solver itself is not presented.

The Siemens Software Simatic Step 7 [Sie08] is able to create communication schedules for PROFINET IRT, but the implemented algorithm has never been published.

IV. Formalisation

In our previous work [GN09] we presented a first formalisation of a PROFINET IRT network. A PROFINET IRT network was described as a directed and weighted graph $G = (V, E, w)$, with V as a set of switches used as vertices and E as the set of network cables connecting the ports of a the switches as edges.

$$G = (V, E, w) \tag{1}$$

$$V = \{s_1, s_2, s_3, ..., s_{|V|}\} \tag{2}$$

$$E = \{e_1, e_2, e_3, ..., e_{|E|}\ \} \tag{3}$$

$$w : E \to \mathbb{N} \tag{4}$$

The function w provides the weight of a given edge. The weight of an edge represents the transmission delay between two switches. This delay is the sum of the bridging and the sending delay (tx–delay) of the sending switch, the delay over the network cable and the receiving–delay (rx–delay) of the receiving switch.

Further definitions for a *path through the network*, a *schedule* and an *optimal schedule* and the *quality of a schedule* can be found in [GN09]. But still, this formalisation is not complete. Multicast messages and macro cycles have not been considered so far. Also "Dynamic Frame Packing" [SJW08], an extension of the PROFINET standard, is not part of the formalisation. For a complete model of the communication system, some hardware specific parameters like, for example, the minimum gap between two messages are also necessary.

Within our work we will go one step further to a complete formalisation of PROFINET IRT.

V. Use–Cases

PROFINET IRT can be set up in any network topology. For this reason, following topologies will be considered: line, line with branches, star, circle, full meshed and (partially) meshed. Additionally, the different kinds of communication traffic (uni– and multicast messages), different optimisation goals, message dependencies and combinations of these variables will also be considered as use–cases. For each of these use–cases, the corresponding complexity of the scheduling problem is analysed.

While the communication planning in the line topology is rather simple, the planning for a partially or fully meshed network is far more difficult. In our work we will classify the different use–cases in complexity classes. This can be seen as a pioneering work to find a suitable algorithm for each use–case.

VI. Summary and Outlook

Current communication systems in industrial automation are often based on Ethernet. This allows for vertical integration. For example, (in theory) it is possible for an office PC to communicate with an i/o–device on the factory floor. But to comply with the real–time requirements of factory automation it is necessary, among other things, to plan the communication in advance. The complexity of this planning task depends directly on the network topology.

The contribution of our work will be a formal definition of the communication system PROFINET IRT. Furthermore, the different network topologies and the planning (scheduling) of communication in these networks will be divided into use–cases which will be investigated with respect to the complexity of the necessary planning task.

References

[Dop08] F. Dopatka. *Ein Framework für echtzeitfähige Ethernet-Netzwerke in der Automatisierungstechnik mit variabler Kompatibilität zu Standard–Ethernet*. PhD thesis, Fachbereich Elekrotechnik und Informatik der Universität Siegen, 2008.

[Fel00] M. Felser. Ethernet als Feldbus? Kommunikationsmodelle für industrielle Netzwerke, 2000. http://felser.ch/download/FE-TR-0005.PDF (09.01.2009).

[GN09] O. Graeser and O. Niggemann. Planning of time triggered communication schedules. In W. A. Halang and P. Holleczek, editors, *Software–intensive verteile Echtzeitsysteme*, Informatik aktuell, Berlin Heidelberg, 2009. Springer.

[HBv09] Z. Hanzalek, P. Burget, and P. Šucha. Profinet io irt message scheduling. In *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 57–65, Washington, DC, USA, 2009. IEEE Computer Society.

[Jan02] D. Janssen. Ethernet–Kommunikation in Echtzeit — Echtzeit ohne Isolation, 2002. www.plastverarbeiter.de/ai/resources/e0aed1e95ba.pdf (January 2009).

[Jas05] Jürgen Jasperneite. Echtzeit–Ethernet im Überblick. *atp — Automatisierungstechnische Praxis*, 3, März 2005.

[JB03] D. Jansen and H. Büttner. Ethercat — der ethernet–feldbus, teil 1: Funktionsweise und eigenschaften. *Elektronik*, (23):62–67, 2003.

[Ope10] Open DeviceNet Vendors Association. Ethernet/ip, 2010. http://www.ethernetip.de (April 2010).

[Pop05] M. Popp. *Das Profinet IO–Buch — Grundlagen und Tipps für Anwender*. Hüthing–Verlag, Heidelberg, 2005.

[SER08] SERCOS International. Funktionsweise SERCOS III, Januar 2008. http://www.sercos.de/Funktionsweise.179.0.html (January 2008).

[Sie08] Siemens. *Simatic Manager*. Technical Documentation, 2008.

[SJW08] M. Schumacher, J. Jasperneite, and K. Weber. A new approach for increasing the performance of the industrial ethernet system profinet. In *7th IEEE International Workshop on Factory Communication Systems (WFCS 2008)*, pages 159 – 167, Dresden, Germany, May 2008.

Real-Time Analysis of Round-based Distributed Algorithms

Alexander Kößler, Heinrich Moser, Ulrich Schmid
Embedded Computing Systems Group (E182/2)
Technische Universität Wien, 1040 Vienna, Austria
{koe,moser,s}@ecs.tuwien.ac.at

I. Distributed Computing vs. Real-Time Systems Research

Designing sound fault-tolerant distributed real-time systems requires a scientific basis, which allows to cope with three very different and partially conflicting major issues:

(SD) *Spatial distribution*, i.e., multiple processors, typically coupled via some network(s), executing multiple processes that are working towards a common goal.

(PF) *Partial failures* of system components, which may fail independently and without immediate recognition of each other.

(RT) *Real-time requirements*, put on the response times of certain events by the environment. Additional concerns may be low power consumption, costs, etc.

The major consequence of (SD) and (PF) is *uncertainty* of the local processes about the global system state: One never knows exactly how far the execution of other processes proceeded and how long it takes for a message to arrive. Fault-tolerant distributed algorithms [1] have been invented to cope with this uncertainty.

Unfortunately, however, distributed algorithms research usually ignores real-time aspects: First, the wealth of research on (lock-step) *synchronous* systems assumes that all processes are perfectly synchronized by means of a common clock, and that all messages sent in step k are received by step $k+1$. This is a very convenient restriction, as it rules out any uncertainty due to asynchrony and leaves only uncertainty due to failures; application-level modeling, (real-time) analysis and programming are hence easy. At the same time, however, it makes the system critically dependent on the continuous maintenance of synchronization: A synchronous system may even lose untimed safety properties like consistency of replicated data—not just timeliness properties—if synchronization is lost. Moreover, *implementing* synchrony (e.g. by means of a distributed clock synchronization algorithm [2], [3]) is costly and requires a priori bounds on end-to-end message delays.

Asynchronous distributed algorithms do not suffer from such problems. Unfortunately, however, modeling, (real-time) analysis, and programming are considerably more involved: (1) Most important distributed computing problems, like consensus, are impossible to solve in purely asynchronous systems in the presence of failures [4] (which makes *partially synchronous* (ParSync) systems like [5], [6], [7] attractive). (2) The convenient time-triggered (periodic) invocation of processes, which is a prerequisite for all existing real-time scheduling approaches, is replaced by a message-driven invocation. (3) Timing aspects are usually abstracted away: Processes are modeled as state machines, which perform *zero-time* computing steps; time can only be modeled via the interval between computing steps here. Hence, the issues of queueing effects and scheduling do not arise at all.

As a consequence, the time complexity results obtained in distributed algorithms research are typically not particularly meaningful for real systems, and sometimes even too optimistic [8]. Bridging the gap between distributed algorithms and real-time systems research requires new approaches and analysis methods.

II. The Real-Time Distributed Computing Model

We developed a *real-time distributed computing model* (RT model) [9], [10], [11], which replaces the zero-time steps of "classic" distributed computing models by non-zero-time *jobs*, shown in Figure 1, and hence allows to deal explicitly with queueing and scheduling. Most importantly, it allows to remove the classic assumption of *a priori* given end-to-end message delay bounds Δ, which are just considered as *model* parameters in classic distributed algorithms research. In reality, however, Δ not only depends on "raw" system parameters like computing step times μ and transmission delays δ, but also on the load (message transmissions, computations) created by a distributed algorithm $\mathcal{A}$, and the scheduling disciplines $\mathcal{S}$ employed for processes and messages. Hence, $\Delta = F(\mathcal{A}, \delta, \mu, \mathcal{S})$ for some function F. On the other hand, the distributed algorithm $\mathcal{A}$ itself may of course depend on Δ, just recall the round duration of a synchronous algorithm or the need for setting message time-outs. As a consequence, we usually have $\mathcal{A} = D(\Delta)$ for some function D. This creates a cyclic dependency of the algorithm and the end-to-end delays as shown in Figure 2, i.e., $\Delta = F(\mathcal{A}, \delta, \mu, \mathcal{S}) = F\big(D(\Delta), \delta, \mu, \mathcal{S}\big)$.

To break the cyclic dependency, the "real" equation $\Delta = F\big(D(\Delta), \delta, \mu, \mathcal{S}\big)$ must be derived and solved by means of a detailed real-time analysis. This analysis must incorporate the generated system load, the scheduling disciplines for processors and network, and of course the raw computing and transmission delays in order to derive a bound Δ that indeed holds when algorithm $\mathcal{A}$ is executed in a given system. This

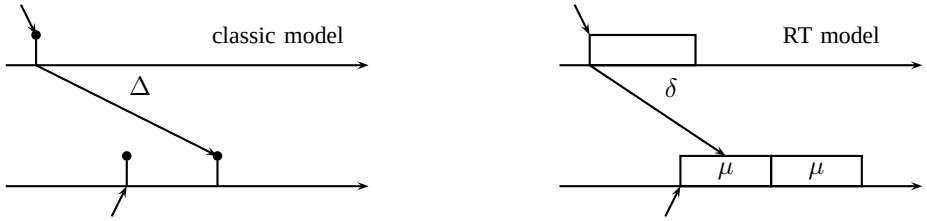


Figure 1. Comparison between the "classic" distributed computing model and "our" RT Model

paper is devoted to some preliminary results of our research on this problem.

III. Round Duration of a Synchronous Algorithm: A Simple Example

We examined a simple "oral messages" Byzantine Generals algorithm [12], running on n processors in a completely synchronous lock-step round-based system. This algorithm works as follows: The *commander* (= one designated processor) broadcasts some initial value. Afterwards, each round basically consists of echoing all information heard so far, to ensure that, after $f+1$ rounds, all (non-faulty) processors have enough information to decide on a common value, despite the fact that $f < n/3$ of these processors (including the commander) might be faulty and send out incorrect data.

Although this kind of "full information exchange" is a common pattern in fault-tolerant distributed algorithms, it causes the amount of data exchanged to increase exponentially with each round (reflected by an increasing number of messages). As it turns out [13], when using an optimal scheduling policy, the worst-case duration of a fault-free round for this algorithm is exactly $W = \max\{W^a, W^b, W^c\}$, with

$$\begin{array}{lllllll} W^a = C & + & S \cdot \#_S & & & + & R \cdot \#_R, \\ W^b = C & & & + & \delta^+ & + & R \cdot \#_R, \\ W^c = C & + & S \cdot (\#_S - 1) & + & \delta^+ & + & R \cdot (n - r - 1), \end{array}$$

representing three possible critical paths. Herein, $C/S/R$ is the worst-case execution time of a computation/sending/receiving job, δ^+ is an upper bound on the message transmission delay, $\#_S/\#_R$ is the number of send/receive jobs in the current round, and r is the round number.

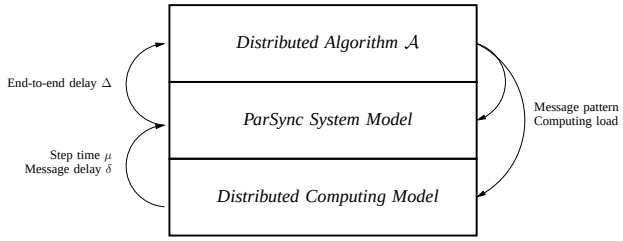


Figure 2. Dependency of the distributed computing model, the system model and a distributed algorithm.

IV. Round Duration without Lock-Step Synchrony: An Open Problem

In the above "toy example", all rounds start (periodically) in perfect synchrony. This convenient feature is lost in close-to-asynchronous ParSync algorithms, where some round $r+1$ is started upon certain trigger events, like the arrival of the $n - f$-th round r message from different processes. Since the latters' round r starting times are not synchronized and the end-to-end delays may vary considerably, the round $r+1$ starting times of different processors are typically even further apart. Keeping track of the resulting non-linear mutual depencencies requires a powerful mathematical method. Two promising approaches are illustrated below:

A. Max-Plus Algebra

When analyzing the time complexity of (non-fault-tolerant) asynchronous algorithms, one typically observes a certain structure of the resulting expressions, consisting of $\max$ and $+$ operations only. To analyze such equations, there exists a well studied algebraic framework, namely, the Max-Plus algebra $\mathcal{R}_{max} = (\mathbb{R}_{max}, \oplus, \otimes, \epsilon, e)$ where $\epsilon := -\infty$, $e := 0$, and $\mathbb{R}_{max} := \mathbb{R} \cup \epsilon$. For elements $a, b \in \mathbb{R}_{max}$, the operators are defined as $a \oplus b := \max(a, b)$ and $a \otimes b := a + b$ [14]. To illustrate its use, consider the following simple distributed round synchronizer algorithm for the classic zero-step-time model: After initialization, every process (out of n) sends its round 1 message to every other process. If a process received all $n - 1$ round k messages, it switches to round $k + 1$ and broadcasts its round $k + 1$ message. The time series $x_i(k)$, $k \geq 1$, of every process i's round switching times, and derived quantities like the average round duration up to round k, can be calculated as follows: Given a transmission delay matrix $[\delta] \in \mathbb{R}^{n \times n}$ where $[\delta]_{ij} = \delta_{ij}$ is the delay of a message from process i to process j, we obtain the recursive formula $x_i(k) = \bigoplus_{j=1}^{n} x_j(k-1) + \delta_{ji}$. Using Max-Plus matrix multiplication with the delay matrix $[\delta]$ and introducing the round switching time vector $\overrightarrow{X}(k) \in \mathbb{R}^n_{max} := (x_1(k), x_2(k), \ldots, x_n(k))^T$, this can be expanded to $\overrightarrow{X}(k) = (\delta^T)^{\otimes k} \otimes \overrightarrow{X}(0)$, with $\overrightarrow{X}(0)$ containing the initial starting times of the processes.

B. When Max-Plus is not Enough

Unfortunately, when making the transition to the RT model, where every incoming message triggers a non-zero-time job, queuing effects require an additional operation to be considered in time complexity expressions: the $\min$ operation.

Note that the same is true, even in the classic model, when *fault-tolerant* asynchronous algorithms are considered. Dealing with the resulting equations requires the *Min-Max-Plus algebra* [15], where, in contrast to Max-Plus, not too many general results seem to be available yet.

We believe that combining the RT model and Min-Max-Plus algebra will result in a powerful framework for the analysis of partially synchronous fault-tolerant distributed algorithms. Needless to say, however, there is still a long way to go in order to solve the various research problems that arise in this context.

REFERENCES

[1] N. Lynch, *Distributed Algorithms*. San Francisco, USA: Morgan Kaufman Publishers, Inc., 1996.

[2] B. Simons, J. Lundelius-Welch, and N. Lynch, "An overview of clock synchronization," in *Fault-Tolerant Distributed Computing*, ser. LNCS 448, B. Simons and A. Spector, Eds. Springer Verlag, 1990, pp. 84–96. [Online]. Available: http://faculty.cs.tamu.edu/welch/papers/lncs90.ps

[3] U. Schmid, Ed., *Special Issue on The Challenge of Global Time in Large-Scale Distributed Real-Time Systems*, ser. J. Real-Time Systems 12(1–3), 1997.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[5] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988.

[6] J. Widder and U. Schmid, "The Theta-Model: Achieving synchrony without clocks," *Distributed Computing*, vol. 22, no. 1, pp. 29–47, Apr. 2009.

[7] P. Robinson and U. Schmid, "The Asynchronous Bounded-Cycle Model," in *Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'08)*, ser. Lecture Notes in Computer Science, vol. 5340. Detroit, USA: Springer Verlag, Nov. 2008, pp. 246–262, (Best Paper Award). [Online]. Available: http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=2398%\&viewmode=paper\&year=2008

[8] H. Moser and U. Schmid, "Optimal clock synchronization revisited: Upper and lower bounds in real-time systems," in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, ser. LNCS 4305. Bordeaux & Saint-Emilion, France: Springer Verlag, Dec 2006, pp. 95–109. [Online]. Available: http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=2068%\&viewmode=paper\&year=2006

[9] ——, "Reconciling distributed computing models and real-time systems," in *Proceedings Work in Progress Session of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, Rio de Janeiro, Brazil, Dec 2006, pp. 73–76. [Online]. Available: http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=2055%\&viewmode=paper\&year=2006

[10] H. Moser, "Towards a real-time distributed computing model," *Theoretical Computer Science*, vol. 410, no. 6–7, pp. 629–659, Feb 2009.

[11] H. Moser and U. Schmid, "Optimal deterministic remote clock estimation in real-time systems," in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, Luxor, Egypt, Dec. 2008, pp. 363–387.

[12] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, July 1982.

[13] H. Moser, "The byzantine generals' round duration," Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-2, 1040 Vienna, Austria, Research Report 9/2010, 2010.

[14] B. Heidergott, G. J. Olsder, and J. von der Woude, *Max plus at work*. Princeton Univ. Press, 2006.

[15] Y. Cheng, "A survey of the theory of min-max systems," in *Springer LNCS 3645 (Advances in Intelligent Computing ICIC'05, Part II)*, 2005, pp. 616–625.

Open Problems in Scheduling Self-Suspending Tasks

Karthik Lakshmanan[†], Shinpei Kato[†‡], and Ragunathan (Raj) Rajkumar[†]
[†]Department of Electrical and Computer Engineering, Carnegie Mellon University
[‡]Department of Computer Science, The University of Tokyo

Abstract

Self-suspension intervals are becoming increasingly common in various systems such as: (i) multi-core processors, where tasks running on one core have to synchronize with tasks running on other cores, (ii) heterogeneous ISA multi-core processors, where certain instructions can only be executed on specific processor cores, and (iii) systems with accelerated co-processors such as Digital Signal Processors (DSPs) or Graphics Processing Units (GPUs). In the light of these developments, a few key questions arise: (a) What should be the standard task model for specifying suspension intervals in such systems? (b) Given that classical scheduling algorithms such as Earliest-Deadline First (EDF) suffer from scheduling anomalies in self-suspending task systems, does there exist a competitive anomaly-free scheduling algorithm for such systems? (c) Given that the feasibility problem of scheduling periodic tasks with at most one self-suspension per task and implicit deadlines is NP-Hard, what can we say about the feasibility condition for scheduling sporadic self-suspending task systems? In this position paper, we provide some preliminary ideas and intuitions towards answering these questions, and seek to engage the broader real-time research community in solving these open problems.

1. Problem Context

Recent years have seen significant changes in the landscape of processor technologies, ranging from the emerging trend of massively multi-core processors to general-purpose computing support in Graphics Processing Units (GPUs). These technology trends project a future in which computation is no longer independently carried out on one processor core but needs to be often synchronized with other processor cores and GPUs. Synchronizing with external events results in suspension intervals, where tasks voluntarily release control of the processor for extended invervals of time. In such systems, classical real-time task models need to be augmented with upper bounds on the duration of suspension intervals. This is an important problem since dealing with suspension intervals as just part of the computation time itself is not a scalable approach as tasks on each processor core are going to increasingly rely on other processor cores and co-processors.

2. Task Models

Developing an useful and practical task model is the first basic requirement for advancing the state of the art in analyzing self-suspending tasks. Traditional approaches have looked at the following task models:

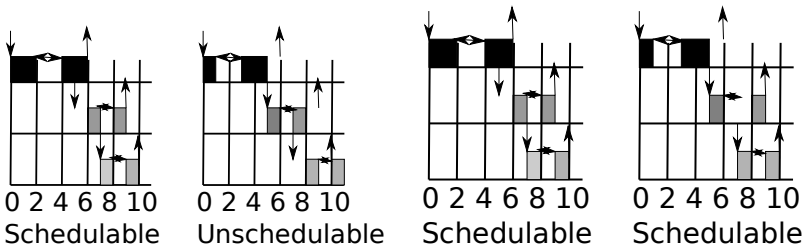
2.1 The $\tau : (C, E, T, D)$ Traditional Model

τ is a task that releases jobs with a minimum inter-arrival time of T time units, each job of τ has a relative deadline of D time units from its release, C is an upper bound on the total execution time of each job of τ, and E is an upper bound on the total suspension time for each job of τ.

From an analysis perspective, this leads to quite pessimistic results since the locations of the suspensions within the jobs of τ are unknown. From a system designer perspective, this is a task model that is easy to specify and reason about. It alleviates the burden of finding the exact number of suspension intervals and their locations with respect to job releases. This approach has been used in work reported in [5, 3, 4].

2.2 The $\tau : ((C^1, E^1, C^2, E^2, ..., C^m), T, D)$ Extended Model

τ is a task that releases jobs with a minimum inter-arrival time of T time units, each job of τ has a relative deadline of D time units from its release. The execution of each job of τ comprises of m computation segments interleaved by $m-1$ suspension intervals. This model subsumes the (C, E, T, D) model in that tasks in the $((C^1, E^1, C^2, E^2, ..., C^m), T, D)$ model can be modeled using (C, E, T, D) as $(\sum_{j=1}^{m} C^j, \sum_{k=1}^{m-1} E^k, T, D)$. This approach has been used in [2, 6].



(a) Anomaly under EDF (b) EDZL Scheduling

Figure 1. Scheduling anomaly with self-suspending tasks under EDF and prevention using EDZL

2.3 Computation Graphs

A more exhaustive approach could be to model the computation of each job as a Directed Acyclic Graph (DAG), which enables the modeling of branches and leads to a more precise representation of the task. Analyzing the schedulability of such DAGs might be more complex but it has the best potential for achieving an exact-case analysis since it subsumes both the (C, E, T, D) and $((C^1, E^1, C^2, E^2, ..., C^m), T, D)$ models.

An open question related to this is the following: *What is a practical and useful model for specifying suspension intervals in real-time tasks?*

3. Scheduling Anomalies from Self-Suspension

Reference [6] showed that scheduling anomalies can arise from using classical scheduling algorithms such as EDF in self-suspending task systems. The example task set used in [6] is given in Figure 1(a), where a job J_1 with execution pattern $(2, 2, 2)$ and deadline 6 is released at time 0, job J_2 with execution pattern $(1, 1, 1)$ and deadline 4 is released at time 5, and job J_3 with execution pattern $(1, 1, 1)$ and deadline 3 is released at time 7. Reducing the execution-time requirement of the first segment of J_1 by 1 time unit causes J_3 to miss its deadline under EDF, as shown in Figure 1(a).

An open question is *whether there exists a competitive anomaly-free scheduling algorithm for such task systems with better performance than EDF.*

In terms of answering this question, it is known that *EDZL* (Earliest-Deadline First with Zero-Laxity) [1] performs better than EDF on *non-suspending* tasks. An interesting point to start would be applying *EDZL* (Earliest-Deadline First with Zero-Laxity) [1] scheduling to *self-suspending* tasks. We show in Figure 1(b) that EDZL can avoid the anomaly introduced in EDF.

An open question related to this is *Does dynamic-priority scheduling strictly outperform fixed-priority scheduling in self-suspending task systems?*

4. Self-Suspending Sporadic Task Systems

It is known from [6] that the feasibility problem of scheduling periodic tasks with at most one self-suspension per task and implicit deadlines is NP-hard in the strong sense. However, it remains to be seen whether a polynomial time or pseudo-polynomial time exact-case feasibility condition can be developed for sporadic self-suspending task systems, due to their potential for having a more simpler characterization of the worst case.

An open question is *whether we can develop an anomaly-free exact feasibility condition for self-suspending sporadic task systems*.

In this regard, with respect to fixed-priority scheduling, [2] provides an exact characterization of the critical scheduling instant for a self-suspending task with respect to interference from higher-priority non-suspending sporadic tasks.

References

[1] T. P. Baker, M. Cirinei, and M. Bertogna. Edzl scheduling analysis. *Real-Time Syst.*, 40(3):264–289, 2008.

[2] K. Lakshmanan and R. R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *RTAS '10: Proceedings of RTAS 2010*, 2010.

[3] C. Liu and J. H. Anderson. Supporting sporadic pipelined tasks with early-releasing in soft real-time multiprocessor systems. In *RTCSA '09: Proceedings of the 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 284–293, Washington, DC, USA, 2009. IEEE Computer Society.

[4] C. Liu and J. H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *RTSS '09: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, pages 425–436, Washington, DC, USA, 2009. IEEE Computer Society.

[5] J. Liu. In *Real-Time Systems*. prentice hall, 2000.

[6] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium*, pages 47–56, Washington, DC, USA, 2004. IEEE Computer Society.

Scheduling of self-suspending tasks: state of art and new insights

Frédéric Ridouard, Pascal Richard
LISI-ENSMA and University of Poitiers
{frederic.ridouard,pascal.richard}@ensma.fr

1 Introduction

Most of real-time systems contain tasks with self-suspension. A task with a self-suspension is a task that during its execution prepares specifics computations (e.g. In/Out operations or *FFT* on a digital signal processor). The task is self-suspended to execute the specifics computations upon external dedicated processors. External operations introduce self-suspension delays in the behavior of tasks. The task waits until the completion of the external operations to finish its execution. Generally, the execution requirement of external operations can be integrated in the execution requirement of the task. But, if self-suspension delays are large, then such an approach cannot be used to achieve a schedulable system. Thus, self-suspension must be explicitly considered in the task model.
We consider sporadic tasks with self-suspension. Let I be a task system of n tasks. Every occurrence of a task is called a *job*. Every task τ_i ($1 \leq i \leq n$) arrives in the system at time 0, its relative deadline is denoted D_i and its period T_i. We assume that tasks are subjected to constrained-deadlines ($D_i \leq T_i$). Tasks are allowed to self-suspend at most once. Every task τ_i ($1 \leq i \leq n$) has two subtasks (with a maximum execution requirement $C_{i,k}$, $1 \leq k \leq 2$) separated by a maximum self-suspension delay X_i between the completion of the first subtask and the start of the second subtask. Such delays change from one execution to another since they model execution requirements of external operations. Consequently every task τ_i is denoted: $\tau_i : (C_{i,1}, X_i, C_{i,2}, D_i, T_i)$.
Only few positive results have been defined for schedulability analysis of self-suspending tasks. Most of them exploit particular task sets and restrictive assumptions. But to the best of our knowledge, no solution provides simultaneously results for efficiently scheduling and analysing self-suspending tasks. In this note, we present the state of the art about the self-suspending tasks. Then, we present some possible insights to define solutions to this scheduling problem.

2 Known results

It has been already proved in [9, 8] that the feasibility problem of scheduling self-suspending task systems is $\mathcal{NP}$-Hard in the strong sense. We have also shown the presence of scheduling anomalies under fixed priorities and *EDF* for scheduling independent tasks with self-suspension upon an uniprocessor platform when preemption is allowed. We have also proved that classical on-line scheduling algorithms (*EDF*,*LLF*,*RM*,*DM*) are not better than 2-competitive to minimize the maximum response time and not competitive to minimize the number of tardy tasks. Response Time Analysis for fixed-priority scheduling algorithm *RM*, [2, 6] have been proposed for computing response time upper bounds. Finally, we have also shown that it is impossible to define an optimal on-line algorithm to schedule sporadic tasks systems when tasks are allowed to self-suspend (cf. [8]).
In [3], the authors characterize the exact critical instant for self-suspending sporadic tasks. They deduce a pseudo-polynomial response-time tests for analysing the schedulability of such self-suspending tasks. In [5], the periodic tasksets with suspensions, pipelines, and non-preemptive sections are considered. The authors show how to transform such a task system into a periodic taskset with only suspensions. Then, they use prior results [4] to derive tardiness bounds for more complex systems.

From a practical scheduling point of view, in [1] is presented a configurable synchronization protocol for self-supending process sets. In fact, the protocol extends the concept of priority ceilings. Furthermore, an algorithm for computing the corresponding maximum blocking times is presented.

3 New insights

We next present some ideas to develop in order to achieve valuable positive results on scheduling self-suspending tasks:

Particular task characteristics. Most of negative results are based on instance problems in which suspension delays are quite huge. A recurrent problem with the scheduling of self-suspending tasks is the duration of suspension delays since it can be so important that the computation time is negligible. Studying particular task set, while fixing some properties of task parameters usually help to understand the difficulties encountered while solving a general complex problem. For that purpose, we think that it is important to investigate some particular cases such as: suspension delays cannot exceed processing time for each task, suspension delays are all equal a constant, suspension delays are all equal to 1, etc. It is surely one way to achieve some positive results.

Resource augmentation technique. Classical scheduling algorithms are not optimal for scheduling tasks allowed to self-suspend. An important question is: is there a processor speed s so that a classical scheduling algorithm (e.g., *RM* or *EDF*) will lead to a feasible schedule if one exists upon a unit-speed processor (i.e., computed by an optimal off-line scheduling algorithm).

Impact of scheduling anomalies. What is the impact of considering only worst-case execution time and suspension delays while performing a schedulability analysis? Is-it possible to establish simular results as these one obtained by Mok et al. in [7] that analyses the robustness of non-preemptive scheduling for *RM* and *EDF*. They proved that scheduling anomalies can lead to miss at most 50% of deadlines. If such a positive result cannot be achieved for the general self-suspending task scheduling problems, but may be some basic assumptions on task parameters will help (e.g., bounding the ratio $\frac{X_i}{C_i}$) to formulate sufficient conditions for the self-suspending robustness.

The multiprocessor point of view. Self-suspending taskset can in fact be modelled by chains of tasks, where suspension delays are tasks run upon some dedicated processors (e.g. I/O processing devices). We think that known results in the multiprocessor scheduling theory can be used to derive some results for uni-processor scheduling of self-suspending tasks.

References

[1] Y.S. Chen and L.P. Chang. A real-time configurable synchronization protocol for self-suspending process sets. *Real-Time Systems*, 42:34–62, 2009.

[2] I-G. Kim, K-H. Choi, S-K. Park, D-Y. Kim, and M-P. Hong. Real-time scheduling of tasks that contain the external blocking intervals. *Real-Time and Embedded Computing Systems and Applications(RTCSA'95)*, 1995.

[3] K. Lakshmanan and R. (Raj) Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10), Stockholm, Sweden*, (12–15), April 2010.

[4] C. Liu and J.H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. *Proceedings of the 30th IEEE Real-Time System Symposium*, pages 425–436, 2009.

[5] C. Liu and J.H. Anderson. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems. *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10), Stockholm, Sweden*, (12–15), April 2010.

[6] Jane W. S. Liu. *Real-Time Systems*, chapter Priority-Driven Scheduling of Periodics Tasks, pages 164–165. Prentice Hall, 2000.

[7] A.K. Mok and W.C. Poon. Non-preemptive robustness under reduced system load. *Proceedings of the 26th IEEE Real-Time System Symposium (RTSS'05)*, 2005.

[8] F. Ridouard and P. Richard. Worst-case analysis of feasibility tests for self-suspending tasks. In *proc. 14th Real-Time and Network Systems, Poitiers*, 2006.

[9] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, 1, December 2004.

Efficient RMS schedulability tests

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

28 May, 2010

1 Introduction

Priority-based scheduling branches into static and dynamic priority scheduling. The optimal static priority algorithm is Rate-Monotonic Scheduling (RMS, for the case of implicit deadlines). In the dynamic priority case where priorities are no longer attached to an entire task, Earliest-Deadline First (EDF) and Least-Laxity First (LLF) are optimal algorithms. While an exact schedulability test both for EDF and LLF is very simple ($u_{total} \leq 1$), exact tests for RMS are much more complex. Time-Demand Analysis (TDA) [7] is, besides simulation along the entire hyperperiod, a solution, but one with pseudo-polynomial complexity.[1] Thus, several only *sufficient* tests with linear, log-linear, quadratic or cubic complexity have been suggested.

In [8], we have recommended the increased use of the sufficient RMS schedulability tests *Distance-Constrained Tasks* (DCT) and *Specialization with respect to r* (Sr), first given by Han and Tyan [2][3]. Related to their medium computational complexity of $\mathcal{O}(n^2)$ and $\mathcal{O}(n\ log\ n)$, their performance is very good. They outperform established ones of comparable computational complexity and both are based on the concept of *accelerated simply periodic task sets*. This crystal-clear principle plays a central role in the field of RMS schedulability tests [8].

2 Open Problem 1: DCT/Sr for MP Scheduling

The two tests DCT and Sr serve as good uniprocessor schedulability tests in partitioned multiprocessor scheduling, see [8]. Thus, a further application to multicore scheduling algorithms - especially global ones - shall be investigated. Preliminary work using task splits can be found in [5] and [4] where *simply periodic task sets* up to 100% utilization are scheduled on multiprocessors, first with a minor dynamic element in [5] and then statically in [4]. The step to include in the approach is the *acceleration*.

[1] This means that TDA requires a polynomial number of operations in terms of the number of tasks which is exponential in the length of the input when coding in a non-trivial, i.e., not in unary, denominational number system.

3 Open Problem 2: Parallelization Potential of DCT/Sr

Compared to some linear complexity algorithms like RBound [6] and Burchard criterion [1], DCT and Sr completely exclude expensive operations like logarithms or powers. They are perfectly suited for parallelization since it is a minimization problem with a moving pivot. Thus, even in this meta perspective, state-of-the-art hardware with multicores should be considered. Because of that, a closer investigation of run-time behavior seems to be promising. Minimzing the run-time of the test is a key factor for its application not only offline but also online.

4 Open Problem 3: DCT/Sr for E-efficient Scheduling

For the emerging field of energy-efficient scheduling, the application of DCT and Sr to Dynamic Voltage Scaling (DVS) as done with the Pillai/Shin (PS) test [9] shall be tried. Although DCT and Sr seem to be superior to PS since they provide guarantees, it is not yet clear whether they are suited for DVS. What makes them attractive is the same (DCT) or even better (Sr) computational complexity compared to the Pillai/Shin test. Quick scheduling is of key importance in online scheduling and in DVS since a too long overhead could easily eat up some of the gain.

References

[1] A. Burchard, J. Liebeherr, Yingfeng Oh, and S.H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *Computers, IEEE Transactions on*, 44(12):1429–1442, Dec 1995.

[2] C.-C. Han and H.-Y. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. *Real-Time Systems Symposium, IEEE International*, 0:36–45, 1997.

[3] Ching-Chih Han, Kwei-Jay Lin, and Chao-Ju Hou. Distance-constrained scheduling and its applications to real-time systems. *Computers, IEEE Transactions on*, 45(7):814–826, Jul 1996.

[4] Myoung-Jo Jung, Yeong Rak Seong, and Cheol-Hoon Lee. Optimal RM scheduling for simply periodic tasks on uniform multiprocessors. In *ICHIT '09: Proc. of the 2009 Int'l Conf. on Hybrid Information Technology*, pages 383–389, New York, NY, USA, 2009. ACM.

[5] Shinpei Kato. *REAL-TIME SCHEDULING OF PERIODIC AND APERIODIC TASKS ON MULTIPROCESSOR SYSTEMS*. PhD thesis, Keio University, 2008.

[6] Sylvain Lauzac, Rami Melhem, and Daniel Mossé. An improved rate-monotonic admission control and its applications. *IEEE Trans. Comput.*, 52(3):337–350, 2003.

[7] John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS*, pages 166–171, 1989.

[8] Dirk Müller. Accelerated simply periodic task sets for RM scheduling. In *Proc. of Embedded Real Time Software and Systems (ERTS^2)*, page 46, 2010.

[9] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35(5):89–102, 2001.

A note on task-parallelism upon multiprocessors

Joël Goossens
Université Libre de Bruxelles
Brussels, Belgium
Email: joel.goossens@ulb.ac.be

Shelby Funk
University of Georgia
Athens, GA, USA
Email: shelby@cs.uga.edu

We consider the scheduling of arbitrary deadline periodic task systems on multiprocessors. When the deadline of a task can exceed its period, it is possible that a task may have several jobs be *simultaneously active*. For *uni*processor systems these jobs are serialized (i.e., scheduled in FIFO order). For multiprocessors, we may be able to execute them simultaneously on different processors. Often, the jobs of all tasks are serialized without discussion (see, e.g., [1], [2], [3]). This type of execution would be required if each of a given task's jobs uses data generated by the predecessor jobs. If, on the other hand, each job uses data gathered from environmental sensors, then multiple jobs generated by a single task may be able to execute simultaneously without conflict.

The conventional belief is that disallowing task parallelism is "worse" than allowing it. Below, we demonstrate that there are scenarios in which serialization and simultaneous execution are incomparable scheduling paradigms. In particular, we demonstrate these paradigms are incomparable for systems satisfying the following description:

- All tasks are synchronous and strictly periodic – i.e., a task τ_i with period T_i will generate jobs at times $k \cdot T_i$, for $k = 0, 1, 2, \ldots$.
- The tasks are scheduled using either the Earliest Deadline First (EDF) algorithm or the Deadline Monotonic (DM) algorithm. EDF gives higher priority to jobs with earlier deadlines[1]. DM gives higher priority to jobs generated by tasks with smaller relative deadlines.

While these assumptions are not uncommon, there are many other types of systems. For example, we might want to consider other scheduling algorithms or the more relaxed sporadic task model, in which the period T_i indicates the minimum amount of time between the releases of τ_i's consecutive jobs.

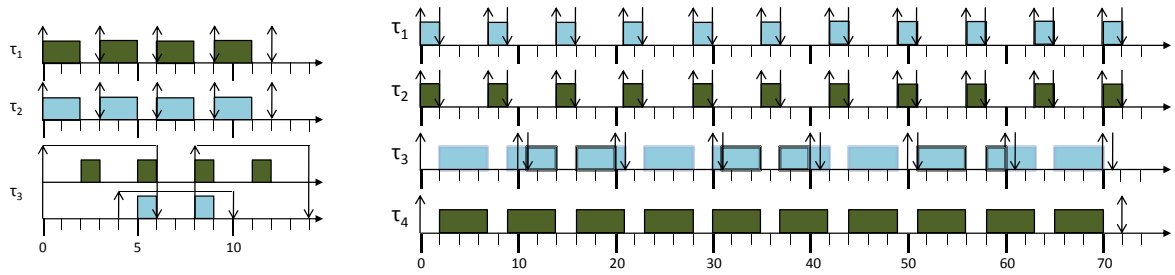
Given that allowing and prohibiting task parallelism are incomparable scheduling variants for the specific scenarios described above, we must consider a number of questions. Specifically,

- Under what conditions does allowing / forbidding task parallelism improve schedulability?
- Are there specific analysis techniques that can be employed when task parallelism is permitted?
- Do these scheduling paradigms remain incomparable when using the sporadic task model?
- Do these paradigms continue to be incomparable for other scheduling algorithms such as LLREF [4], BF [5] or DP-Wrap [6] or the Pfair [7] family of algorithms?

We now demonstrate that these two scheduling variants are incomparable. We begin by defining the terms we will use in the subsequent discussion.

Model and definitions. A periodic or sporadic task set is denoted $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is described using the 3-tuple (T_i, C_i, D_i), where T_i, C_i and D_i are τ_i's period, execution time and relative deadline, respectively. The jobs of a periodic tasks arrive exactly T_i time units apart, with the first job arriving at time 0. The jobs of a sporadic task arrive *at least* T_i time units apart, with the first job arriving no earlier than time 0. If a job of periodic or sporadic task τ_i arrives at time t, it must be allowed to execute for C_i time units during the interval $[t, t + D_i)$. In the special case where the relative deadline is the same as the period, we describe τ_i using the pair (T_i, C_i). We say *task (job) parallelism* occurs when a task (job) runs simultaneously on several processors. Assuming job parallelism is prohibited, we explore the implications of allowing or prohibiting task parallelism.

[1]Throughout this discussion, we assume deadline ties are broken in favor of lower-indexed tasks.



(a) Task parallelism required.

(b) Sequential execution required.

Fig. 1. Allowing and forbidding task parallelism are incomparable paradigms.

Theorem 1. *Allowing / disallowing task parallelism are incomparable variants for both the EDF and DM scheduling of periodic task sets. Specifically,*

- *There are periodic task systems that are schedulable using EDF and DM when task parallelism is allowed, but are unschedulable using these algorithms when task parallelism is forbidden, and*
- *There are periodic task systems that are schedulable using EDF and DM when task parallelism is forbidden, but are unschedulable using these algorithms when task parallelism is allowed.*

Proof: We show each case separately. First, consider the task set $\tau = \{(3,2),(3,2),(4,2,6)\}$. Figure 1(a) illustrates a schedule of this task set upon two processors for 12 time units (the hyperperiod of the system). The color of the tasks indicates which processor the job executes upon. The schedule in Figure 1(a) arises when using either the EDF or the DM scheduling policies. The schedule meets all deadlines *only* because the second job of task τ_3 is allowed to execute simultaneously with the first and third jobs. If parallelism were forbidden for τ_3 then its second job would clearly miss its deadline.

The theorem's second statement is less intuitive. Even so, we show that this statement also holds. Consider the task set $\tau = \{(7,2,2),(7,2,2),(10,7,11),(72,50)\}$. Figure 1(b) illustrates a schedule of this task set upon two processors. As in Figuire 1(a), the color of the tasks indicates which processor the job executes upon and the figure illustrates the schedule generated by both EDF and DM. However, in Figure 1(b) task parallelism is forbidden. Because τ_3's jobs overlap, its jobs have alternating light and dark outlines.

When parallelism is forbidden, the tasks never migrate. Tasks τ_1 and τ_3 execute on processor π_1, and tasks τ_2 and τ_4 execute on processor π_2. Observe that τ_4 completes execution at time 70 and τ_2 must execute during the interval $[70, 72)$. Therefore, if τ_3 ever executes simultaneously on both processors then τ_4 will miss a deadline. When task parallelism is permitted, τ_3 would occupy both processors during the intervals $[10, 11)$ and $[30, 31)$, which would cause τ_4 to miss its first deadline. Observe that in Figure 1(b), all tasks meet their deadlines during the interval $[0, 72)$. Because the tasks are effectively partitioned, this interval presents the worst case scenario for this system. ■

Consequently our intuition is incorrect in the sense that allowing/disallowing task-parallelism are incomparable scheduling variants – at least for EDF and for DM. This observation opens new questions about the multiprocessor scheduling of tasks with unconstrained deadlines. A few of these questions are listed above. We believe additional questions will follow as we gain deeper understanding of these two scheduling paradigms.

REFERENCES

[1] T. P. Baker and M. Cirinei, "A unified analysis of global EDF and fixed-task-priority schedulability of sporadic task systems on multiprocessors," *Embedded Computing*, 2007.

[2] T. P. Baker, "An analysis of EDF schedulability on a multiprocessor," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 8, pp. 760–768, 2005.

[3] S. Baruah and N. Fisher, "Global fixed-priority scheduling of arbitrary-deadline sporadic task systems," *Lecture Notes in Computer Science*, vol. 4904/2008, pp. 215–226, 2008.

[4] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *The Real-Time System Symposium (RTSS)*, Los Alamitos, CA, USA, 2006.

[5] D. Zhu, D. Mosse, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?" in *Real-Time Systems Symposium (RTSS)*, Cancun, Mexico, 2003.

[6] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *EuroMicro Conference on Real-Time Systems (ECRTS)*, Brussels, Belgium, 2010.

[7] S. K. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, June 1996.

Improvement of schedulability bound by task splitting in partitioning scheduling

Frédéric Fauberteau
Université Paris-Est
LIGM, UMR CNRS 8049
Email: fauberte@univ-mlv.fr

Serge Midonnet
Université Paris-Est
LIGM, UMR CNRS 8049
Email: midonnet@univ-mlv.fr

Laurent George
ECE, LACSC
37, quai de Grenelle,
75015 Paris, France
Email: lgeorge@ieee.org

Abstract

We focus on the class of static-priority partitioning scheduling algorithm on multiprocessor. We are interested in improving the schedulability of these algorithms by splitting the tasks which cannot be successfully allocated on processors.

1. Context

Unless $P = NP$, no polynomial time algorithm exists to solve the MULTIPROCESSOR-TASK-PARTITIONING problem. Indeed, this problem can be transformed from BIN-PACKING problem which is NP-hard in the strong sense. Fortunately, there are several heuristics which solve BIN-PACKING problem and they may be adapted to produce partitioned schedulings. For instance, the algorithm FBB-FDD is based on *First-Fit Decreasing* [1] and the algorithm RM-DU-NFS is based on *Next-Fit* [2].

We are interested in proposing a partitioning algorithm which is robust to task Worst Case Execution Time (WCET) overruns faults with the same objective as in [3]. This partitioning tends to maximize the capacity of the system to handle the WCET overruns by offering the maximum *Allowance* to each faulty task of the system. The *Allowance* of a task is the execution duration which can be added to its WCET such that all the deadlines in the system are met [4].

We have shown in [5] that the *Worst-Fit* heuristic offers good results in order to maximize the robustness of a partitioned system. *Worst-Fit* selects the least loaded processor (in terms of utilization) which can accept a task. Unfortunately, the performances of *Worst-Fit* in terms of schedulability are less efficient than those of the heuristics widely used in partitioning algorithms (*First-Fit Decreasing*/*Best-Fit*/*Next-Fit*). In order to improve the schedulability bound of *Worst-Fit* and potentially those of the other heuristics, we propose to split the tasks which cannot be allocated one a single processor. Contrary to the portioned scheduling approach proposed in [6] and [7], we don't migrate a job from a processor to another during its execution but we allocate jobs on different processors such that the migrations occur at job boundaries [8].

2. Task splitting approach

Description. Let Π be a multiprocessor composed by m identical processors denoted by $\Pi = \{\pi_1, \ldots, \pi_m\}$. Let τ be a set of tasks made of n periodic (or sporadic) real-time tasks denoted $\tau = \{\tau_1, \ldots, \tau_n\}$. Each task is characterized by its WCET, its deadline and its period (or minimum interarrival time). We consider a partitioning algorithm denoted A which allocates the tasks of τ on Π. A is designed as follows:

- τ is sorted according to a decreasing order policy (for instance *Decreasing Utilization*),
- a heuristic chooses the processor on which a given task should be allocated,
- a schedulability test is used to decide whether the task can be allocated on the processor chosen by the heuristic.

The resulting algorithm A produces a partition of τ on Π. If A fails to allocate the task τ_i, the set of pending tasks $\{\tau_i, \ldots, \tau_n\}$ is unallocated.

In order to schedule these unallocated tasks, two approaches can be considered: portioned scheduling and scheduling at job boundaries (*i.e.* restricted migration). To illustrate the first one, we show in Figure 1 four tasks

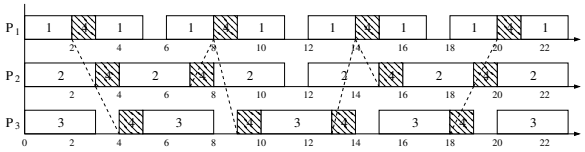


Figure 1. Portioned scheduling.

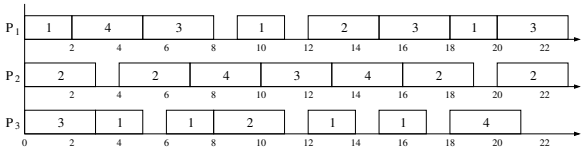


Figure 2. Scheduling at job boundaries.

$\{\tau_1(2,3), \tau_2(3,4), \tau_3(3,5), \tau_4(3,6)\}$ scheduled on three processors. No partition can be found because $u_i + u_j > 1,\ \forall i \neq j,\ i,j \in \{1,2,3,4\}$ ($u_i = \frac{WCET}{period}$). In this example, the tasks are allocated in decreasing utilization order and τ_4 is the last one. Therefore τ_4 is portioned and its jobs are scheduled on the three processors.

This approach can be difficult to implement because it is not easy to split a job in many parts. A job consists in pieces of code which can contain indivisible sections. Moreover, a large number of migrations occurs since a job can migrate several times. The second approach is attractive because the migrations are at job boundaries. Therefore no migration overhead are incurred if the consecutive jobs of a task don't share any data. We propose to plit a task in several k subtasks. Each subtask has the same WCET and deadline as its parent task but the subtasks are less frequently activated (their periods are multiplied). $k-1$ subtasks are unsynchronized with an offset to avoid the jobs overlaps. We represent in Figure 2 the same example as in Figure 1. For instance, the subtasks τ_1^1 (0, 2, 3, 9) on P_1, τ_1^2 (3, 2, 3, 9) on P_3 and τ_1^3 (6, 2, 3, 9) on P_3 produce the same execution as τ_3 (2, 3) (where (w, x, y, z) is (*offset*, *WCET*, *deadline*, *period*)).

Open problem. We consider the case where a partitioning algorithm doesn't succeed to partition the set τ of tasks. Our aim is to build an algorithm which can split the pending tasks in subtasks such that a feasible partition of τ can be found. If no pending task can be split, already allocated tasks can be split until a feasible partition is found. The problem is to build an algorithm which finds a valid splitting scheme if such a scheme exists such that a feasible partition can be found.

Insights. We want to build an algorithm which splits the tasks such that the migration occurs at job boundaries. A first approach is to build a static-priority restricted migration scheduler. This scheduler allocates jobs of the tasks when they are activated on a ready processor according to a static-priority scheme. By logging the allocations made by this scheduler, we can build our splitting scheme. Notice that, this approach can be complex because it may require the consideration of an interval of study which can be exponential. . Moreover, it is not obvious to decide on which processor a task must be activated to minimize the number of subtasks.

References

[1] N. Fisher, S. K. Baruah, and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *Proc. of ECRTS*, 2006, pp. 118–127.

[2] B. Andersson and J. Jonsson, "Preemptive multiprocessor scheduling anomalies," in *Proc. of IPDPS*, 2002, pp. 12–19.

[3] R. I. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *Proc. of RTSS*, 2007, pp. 3–14.

[4] L. Bougueroua, L. George, and S. Midonnet, "Dealing with execution-overruns to improve the temporal robustness of real-time systems scheduled FP and EDF," in *Proc. of ICONS*, 2007, p. 8pp.

[5] F. Fauberteau, S. Midonnet, and L. George, "A robust partitioned scheduling for real-time multiprocessor systems," in *Proc. of DIPES*, September 2010, p. (to appear).

[6] S. Kato and N. Yamasaki, "Portioned static-priority scheduling on multiprocessors," in *Proc. of IPDPS*, 2008, pp. 1–12.

[7] S. Kato and N. Yamasaki, "Semi-partitioned fixed-priority scheduling on multiprocessors," in *Proc. of RTAS*, 2009, pp. 23–32.

[8] S. K. Baruah and J. Carpenter, "Multiprocessor fixed-priority scheduling with restricted interprocessor migrations," in *Proc. of ECRTS*, 2003, pp. 195–202.

Impact of job dropping on the schedulability of uniprocessor probabilistic real-time systems with variable execution times

Olivier Buffet & Liliana Cucu-Grosjean
INRIA Nancy Grand-Est
54600 Villers-lès-Nancy, France
firstname.lastname@loria.fr

***Abstract*—In this paper we address the problem of uniprocessor probabilistic fixed-priority scheduling of real-time systems with variable execution times. For these systems the tasks have an associated probability of missing the deadline, i.e., some jobs may miss their deadlines without affecting the schedulability of the system. Therefore dropping these jobs does not affect the schedulability of the system and it could increase the probability of other jobs to meet their deadline. The problem of deciding what jobs to drop is not trivial and we discuss a possible solution.**

I. Introduction

Requests in real-time environment are often of a recurring nature. Such systems are typically modeled as finite collections of simple, highly repetitive activities (e.g., tasks, messages). When the different instances of those activities are generated in a very predictable manner, we deal with periodic activities. The real-time performances of periodic activities on uniprocessor, distributed or network systems have been extensively studied when all their parameters are known. For some applications the parameters can be unknown until the time instant when the activity is released, or the environment can change forcing the application to adapt. Different approaches can be considered to address these uncertainties (probabilistic approaches, agent systems, learning or game theory, etc) and this paper uses a probabilistic formulation.

II. Statement of the open problem

We deal with the uniprocessor fixed-priority scheduling problem of synchronous periodic tasks with variable execution times. We consider $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ a set of n periodic tasks.

Each task is characterized by an exact inter-arrival time T_i, a relative deadline D_i and a probability of meeting the deadline p_i. It means that the j^{th} activation of τ_i is released at time instant $(j-1)T_i$ and must finish its execution by time instant $(j-1)T_i + D_i$. Among all (representative) activations, at least p_i of them must finish their execution by their deadline. Each activation τ_i has an associated execution time given by a discrete random variable. We denote by $\mathcal{C}_i$ the random variable indicating the execution time of any job of τ_i (see Equation (1)). It is assumed that the random variables giving the execution times are independent.

$$\mathcal{C}_i = \begin{pmatrix} c_k \\ P(C = c_k) \end{pmatrix}_{k \in \{1, \cdots, k_i\}} \quad (1)$$

In Equation (1), $c_k \in [c_i^{min}, c_i^{max}]$ and $k_i \in \mathbb{N}^*$ is the number of values that the random variable $\mathcal{C}_i$ has. We consider that c_i^{min}, c_i^{max} are known.

We denote a task τ_i by $(\mathcal{C}_i, T_i, D_i, p_i)$.

A *schedule* is said *feasible* if any task τ_i has the probability of missing the deadline smaller than p_i.

A schedulability analysis like [1] calculates the response time of a job for the fixed-priority scheduling problem. All jobs released within a hyperperiod contribute to the response time of the corresponding task. Some jobs may always have the obtained response time larger than the deadline, i.e., the probability of missing the deadline is 100%. Therefore we may decide to drop such job before its execution without increasing the probability of missing the deadline for the corresponding task. Nevertheless this mechanism may decrease the probability of missing the deadline for other jobs of the same task or of other tasks.

The problem of deciding what jobs should be dropped in order to obtain a feasible schedule is not trivial. For instance if we decide to drop all jobs that have the probability of missing the deadline of 100%, then we are too pessimistic. After dropping some jobs, lower priority jobs may decrease their probability of missing the deadline and among them, those with the probability originally equal to 100%.

III. Preliminary solution

Intuitively the problem is difficult, even if no such proof exists (to our best knowledge). Therefore we believe that approaches based on systematic search algorithms may propose interesting solutions. In this case (the value of) a solution might be tested using the sum of the probabilities to miss the deadlines of all jobs within a hyperperiod.

References

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

A sharp threshold for rate monotonic schedulability of real-time tasks

Sathish Gopalakrishnan
Department of Electrical and Computer Engineering
The University of British Columbia

I. Setting

For a set of n known task periods $T_1, T_2, \ldots, T_n$, let A_u represent the set of all *implicit-deadline task sets* of utilization u that are schedulable using the rate monotonic scheduling policy. In other words, every task set $\tau \in A_u$ is schedulable using RM and can be represented by a vector of utilizations $\{u_i\}$ such that $0 \leq u_i \leq u$ and $\sum_{i=1}^{n} u_i = u$. For large n and any given set of task periods, there exists a u^* such that, for any $\epsilon, 0 < \epsilon < 1$,

$$\mu(A_u) = \begin{cases} 0 & \text{if } u > (1+\epsilon)u^* \\ 1 & \text{if } u < (1-\epsilon)u^* \end{cases},$$

where $\mu(A_u)$ is the uniform probability measure of the set A_u on the n-dimensional simplex $\sum_{i=1}^{n} u_i = u, u_i \geq 0$ (i.e., the Lebesgue measure of A_u suitably normalized). This result can be obtained by an application of sharp threshold results for random graphs due to Friedgut and Kalai, and Bourgain.

II. Open problem

As $n \to \infty$, for a given set of task periods $T_1, \ldots, T_n$, what is the sharp threshold u^*?

III. Motivation

This problem is interesting for several reasons. An understanding of the measure of RM-schedulable task sets presents us with a much clearer sense of the effectiveness of rate monotonic scheduling. Additionally, in many systems, task periods are fixed and it is useful to understand when a task may be admitted or denied resources. Further, the study of large task sets is interesting because we do not have efficient schedulability tests for this situation. Thus, this direction provides some tools for reasoning about large task sets whereas existing exact tests may be more useful when task sets are small. Lastly, empirical observations indicate that the sharp threshold is higher than known utilization bounds and this allows systems to either admit more tasks or optimize metrics such as energy consumption subject to ensuring that almost all tasks meet their deadlines.

IV. Problem status

Whereas it has been difficult to identify the sharp threshold for rate monotonic scheduling, application of Bourgain's work on sharp thresholds in random graphs [2] allows us to reason about the *existence* of such a threshold for rate monotonic real-time scheduling and other problems [3]. Further, empirical work by several researchers confirms that sharp thresholds exist for multiprocessor scheduling problems [1] as well, although the goal of such empirical work was not to investigate sharp threshold behaviour. The idea of using work from random graphs appears promising and may open up the area for new methods on the average case behaviour or the expected schedulability for real-time scheduling policies. It is also worthwhile noting that the work by Lehoczky, Sha and Ding on breakdown utilization for RM scheduling [4] was intended to capture the average case behaviour of RM scheduling but the notion of breakdown utilization is not as strong as that of a sharp threshold. Furthermore, several approximations were needed to compute the breakdown utilization and the hope is that we can now identify new methods to accurately characterize the probability measure of schedulable (or unschedulable) task sets, thereby gaining a stronger understanding of when task sets are schedulable with high probability. We do, however, know that the relationship between task periods is important. When task periods are harmonic, i.e., of the form $P, kP, k^2P, k^3P, \ldots$, where P is the base period and k is some positive integer, the schedulable utilization bound and the sharp threshold coincide at 1.

References

[1] Brandenburg, B., Calandrino, J., and Anderson, J. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2009), pp. 157–169.

[2] Friedgut, E. Sharp thresholds for graph properties, and the k-SAT problem; with an appendix by Jean Bourgain. *Journal of the American Mathematical Society 12*, 4 (1999), 1017–1054.

[3] Gopalakrishnan, S. Sharp utilization thresholds for some real-time scheduling problems. http://arxiv.org/abs/0912.3852v1, December 2009.

[4] Lehoczky, J. P., Sha, L., and Ding, Y. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium* (1989), pp. 166–171.