ECRTS
10-13 July
Pisa, Italy
2012

RTSOPS 2012

Proceedings of the 3rd International Real-Time Scheduling Open Problems Seminar

Pisa, Italy
July 10, 2012

In conjunction with:
The 24th Euromicro Conference on Real-Time Systems (ECRTS 2012)
July 10-13, 2012

Edited by Liliana Cucu-Grosjean and Nathan Fisher

Foreword

Welcome to Pisa and the 3rd International Real-Time Scheduling Open Problems Seminar (RTSOPS 2012). This seminar provides a venue for the exchange of ideas and 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 cooperation and collaboration within the real-time scheduling community.

This year, we have organized RTSOPS 2012 around joint presentation/collaboration sessions. At the beginning of each session, we will have 2-3 short presentations on a number of important problems in real-time scheduling that give a brief introduction to the problem and its challenges. At the end of each session, we invite the seminar participants (authors and audience members) to interact in small groups, exchanging ideas with the presenters on how the problems might be solved and taking the first steps towards a solution. A total of 11 open problems were selected for presentation at the seminar. These proceedings are also published as an INRIA research report available at hal.inria.fr.

We would like to thank the Steering Committee (listed below) for their hard work in reviewing the open problem submission and helping to make the seminar a success.

Marko Bertogna	University of Modena, Italy
Robert Davis	University of York, UK
Shelby Funk	University of Georgia, USA
Sathish Gopalakrishnan	University of British Columbia, Canada

Special thanks also go to Giorgio Buttazzo and Gerhard Fohler for their support and assistance for organizing this seminar.

Liliana Cucu-Grosjean and Nathan Fisher
Co-Chairs, 3rd International Real-Time Scheduling Open Problems Seminar (RTSOPS 2012)

Table of Contents

The Real-Time Multi-Resource Task Model

Cong Liu
Department of Computer Science, University of North Carolina at Chapel Hill

1 Problem Context

In many real-time and embedded systems, applications utilize multiple resources during execution. As the complexity and the degree of heterogeneity of embedded systems continue to increase, the trend of utilizing multiple system resources, such as CPUs, GPUs, and other special-purpose processors, will continue. For instance, GPGPU (general purpose computing on graphics processing units) and FPGA (field programmable gate array)-based software/hardware co-design are becoming increasingly popular means to assist CPUs in performing complex and intensive computations [3].

To formerly model real-time applications that utilize multiple resources, we present a new real-time sporadic multi-resource (SMR) task model. The SMR task model extends the real-time sporadic task model by allowing tasks to request multiple resources. We assume that there are z resources $\{R_1, R_2, ..., R_z\}$ in the system. We consider the problem of scheduling a set $\tau = \{\tau_1, ..., \tau_n\}$ of n independent SMR tasks on z resources. Each resource R_h ($1 \leq h \leq z$) contains m_h processors. Each SMR task τ_i has at most q_i phases $\{\tau_i^1, ..., \tau_i^{q_i}\}$. Each such phase utilizes a single resource. We assume that any two consecutive phases of any task utilize different resources; otherwise, they can just be viewed as one phase. Also any phase can start execution only if its previous phase (if any) completes. The k^{th} phase of τ_i, τ_i^k, has a worst-case execution time of e_i^k. The worst-case execution time of τ_i on resource R_h (across all of its phases that request this resource) is denoted by $e_i(R_h)$. The worst-case execution time of task τ_i is thus given by $e_i = \sum_{h=1}^{z} e_i(R_h)$ ($e_i(R_h) = 0$ if τ_i does not request resource R_h).

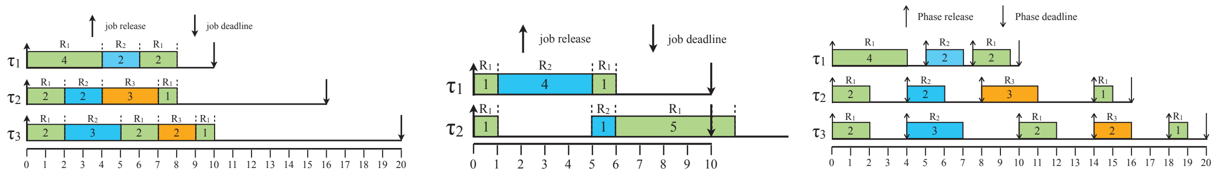
An SMR task is released repeatedly, with each such invocation called a *job*. Successive jobs of the same task are required to execute in sequence. Associated with each task τ_i are a *period* p_i, which specifies the minimum time between two consecutive job releases of τ_i, and a *relative deadline* d_i. For any task T_i, we require $e_i \leq min(d_i, p_i)$. The j^{th} job of T_i, denoted $T_{i,j}$, is released at time $r_{i,j}$ and has a deadline at time $d_{i,j} = r_{i,j} + d_i$. The k^{th} phase of the j^{th} job of τ_i is denoted by $\tau_{i,j}^k$. The *utilization* of an SMR task τ_i on resource R_j is defined by $u_i(R_j) = \frac{e_i(R_j)}{p_i}$. The *total utilization* of task system τ on resource R_j is defined by $u_{sum}(R_j) = \sum_{i=1}^{n} u_i(R_j)$. The *density* of an SMR task τ_i on resource R_j is defined by $\delta_i(R_j) = \frac{e_i(R_j)}{d_i}$.

Non-job-migration and non-preemptivity restrictions on certain resources. Although preemptivity is allowed on some resources such as CPUs, there are resources that only permit non-preemptive executions. For example, execution on GPUs is non-preemtptive [3]. Moreover, job migration, which is allowed under global scheduling approaches on CPUs, may not be efficient on certain resources such as GPUs and FGPAs because significant migration overheads may be incurred. Therefore, to accurately reflect the reality, we allow different scheduling policies, such as preemptive or non-preemptive GEDF, and preemptive or non-preemptive partitioned scheduling, to be used on different resources.

An example SMR task system requesting three resources is given below.

Example 1. *Consider a task set with three SMR tasks τ_1, τ_2, and τ_3 scheduled in a system containing three resources, R_1, R_2, and R_3, as shown in Fig. 1(a). Each resource has two processors. GEDF, preemptive partitioned scheduling, and non-preemptive partitioned scheduling are applied on R_1, R_2, and R_3, respectively. For this task system, $u_{sum}(R_1) = u_1(R_1) + u_2(R_1) + u_3(R_1) = 6/10 + 3/16 + 5/20 = 83/80$. Similarly, $u_{sum}(R_2) = 19/40$ and $u_{sum}(R_3) = 23/80$.*

Related work. A few papers have focused on cooperative scheduling of multiple resources. In [2], the problem of co-scheduling accesses to both a CPU and a disk which are controlled from a single controller was studied. In other work [1], an integrated real-time resource scheduler was designed that performs coordinated allocation and scheduling of multiple resources for periodic soft real-time tasks. However, neither of these papers provide analytically provable timing guarantees as the solutions rely on heuristics, which are purely evaluated based upon run-time performance. Moreover, no formal general real-time multi-resource task model has yet been presented. Therefore, the open problem proposed in this paper is the following: *how to schedule real-time multi-resource tasks on multiprocessors to meet all deadlines*?



(a) (b) (c)

Figure 1: (a) Example SMR task system τ. Phase labels give the execution cost of each phase. τ_1, τ_2, and τ_3 have a period of 10, 16, and 20 time units, respectively. (b) Deadline misses for a lightly-loaded SMR task system that utilizes two resources. (c) Example transformation.

2 A Transformation Approach

We next present an approach that can serve as a first step towards achieving positive results on scheduling SMR task systems. First we show an example unschedulable task system that motivates the proposed approach, as shown in Fig. 1(b). In this example, two tasks τ_1 and τ_2 are executed in a system with two resources, R_1 and R_2. R_1 has two processors and R_2 has one processor. τ_1 first executes on R_1 for one time unit, then on R_2 for four time units, and finally on R_1 for another 1 time unit; τ_2 first executes on R_1 for one time unit, then on R_2 for one time unit, and finally on R_1 for five time units. Both tasks have a period of ten time units. As seen in Fig. 1(b), although this system is lightly loaded ($u_{sum}(R_1) = 0.8$ and $u_{sum}(R_2) = 0.5$), τ_2 still misses its deadline. The observation drawn from this example is that the dependencies among execution phases on different resources plus the interference brought by other tasks quite negatively impact schedulability.

Therefore, our proposed approach is to eliminate such dependencies and interference due to executions on multiple resources. Under our approach, each phase of every MSR task is transformed into a sporadic constrained-deadline subtask that only requests a single resource. This is achieved by assigning phases intermediate release times and deadlines. In order for any MSR task to meet its deadline, we need to guarantee that all phases belonging to this task complete before this deadline. If the last phase $\tau_{i,j}^{q_i}$ of any job $\tau_{i,j}$ of any task τ_i completes by $d_{i,j}$ (which is the original deadline of $\tau_{i,j}$), then $\tau_{i,j}$ meets the deadline as well. The *slack* of each task is distributed among phases to help assign intermediate release times and deadlines to phases. The slack of any task τ_i, denoted by s_i, is defined as $s_i = p_i - e_i$.

Since at any point of time, any task could have at most one active phase (i.e., a phase that has not yet completed and its previous phase, if any, has completed, and the job to which it belongs has been released), the density of any task after transformation is determined by the maximum density among its phases. Thus, one possible way to distribute a task's slack among its phases is to minimize the maximum density among phases. Due to the fact that the maximum of a set of densities is never lower than the average density of the same set, keeping the densities of all phases exactly equal can achieve this goal. By distributing slack in this way, the resulting density of any phase of any task τ_i equals e_i/p_i (derivation details are omitted due to space constraints). Thus, after the transformation, the density of any task τ_i also equals e_i/p_i.

Example 2. *Consider the same task system as shown in Fig. 1(a). Fig. 1(c) shows the transformed task system using our proposed transformation approach. As seen, after the transformation, each phase of each task τ_i has a density of e_i/p_i. For example, for τ_1, each of its phases has a density of 0.8.*

After the transformation, each phase of each SMR task can be treated as an independent (if all assigned deadlines are met) sporadic constrained-deadline task that requests only a single resource. Thus, by applying prior schedulability tests designed for ordinary sporadic constrained-deadline task systems scheduled under preemptive GEDF, or preemptive and non-preemptive partitioned scheduling, we are able to obtain the resulting schedulability tests for the original SMR task system. A quantitative characterization of the worst-case schedulability performance under the proposed approach can also be provided by using the resource augmentation technique.

Other insights. The above transformation approach is clearly pessimistic. In order to better understand the problem and find more efficient solutions, studying particular task sets, where system and task parameters are fixed and simplified, might help. Several particular cases may be worth investigation, for instance, SMR task systems that only request two resources each of which contains a single processor, or SMR task systems in which execution times of all phases of all tasks are identical, or a simplified SMR task model where we assume that there are only two resources in the system and each SMR task contains two phases each of which requests one resource.

References

[1] K. Gopalan and T. Chuieh. Multi-resource allocation and scheduling for periodic soft real-time applications. In *Multimedia Computing and Networking*, pp. 34-45, 2002.

[2] S. Saewong and R. Jajkumar. Cooperative scheduling of multiple resources. In *Proc. of the 20th IEEE Real-Time Systems Symp.*, pp. 90-101, 1999.

[3] R. Wu, B. Zhang, and M. Hsu. GPU-Accelerated Large Scale Analytics. In *HP Labotories Technical Report*, HPL-2009-38, 2009.

Dependent Probabilistic Real-Time

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

Introduction. The real-time community has recently discovered interests in probabilistic approaches to schedulability analysis of time constrained systems. Since the paper of Diaz et al. [1], the probabilistic analysis of real-time systems has been used regularly and with the basic rationale of applying the theory of probability for feasibility analyses [2, 3]. Such analyses are mainly proposed for calculating the response time of tasks [4, 5] under a known preemptive scheduling policy. Based on a rich probabilistic model (rich in terms of associated information), the resulting flexible probabilistic analysis can reduce the pessimism of classical deterministic approaches, refers to [6] as an example.

A *probabilistic real-time system* is intended as a real-time system with at least one parameter described by a random variable. Therefore, a probabilistic real-time analysis consists of the temporal analysis of probabilistic systems relying on probability distributions and cumulative distribution functions. It aims to computing probabilities of events under different conditions (i.e. the response time of a system of tasks), also known as joint probability distributions. The joint probability distribution of two random variables $\mathcal{X}$ and $\mathcal{Y}$ is the probability of event defined in terms of both $\mathcal{X}$ and $\mathcal{Y}$, $\mathbb{P}\{\mathcal{X} = x \text{ and } \mathcal{Y} = y\} = \mathbb{P}\{\mathcal{Y} = y \mid \mathcal{X} = x\} \cdot \mathbb{P}\{\mathcal{X} = x\} = \mathbb{P}\{\mathcal{X} = x \mid \mathcal{Y} = y\} \cdot \mathbb{P}\{\mathcal{Y} = y\}$.

Probabilistic Modeling. A task τ_i is mostly characterized by an offset O_i, a relative deadline D_i, a inter-arrival time (period) T_i and a worst-case execution time C_i. In its probabilistic instance it is possible to have any of its parameters[1] described by random variables. The worst-case execution time can be $\mathcal{C}_i$ as the random variable providing the possible values for worst-case execution time of task τ_i, $\mathcal{C}_i = \begin{pmatrix} C_{i,k} \\ \mathbb{P}(\mathcal{C}_i = C_{i,k}) \end{pmatrix}_{k \in \{1,\cdots,k_{\mathcal{C}_i}\}}$, where $C_{i,k} \in [C_i^{min}, C_i^{max}]$ and $k_{\mathcal{C}_i} \in \mathbb{N}^*$ is the number of values that the random variable $\mathcal{C}_i$ has. Each worst-case execution time value $C_{i,k}$ has a probability associated being the probability of such worst-case happening for any of the task job. A random variable generalizes the notion of deterministic worst-case execution time $\mathcal{C}_i = \begin{pmatrix} C_i \\ 1 \end{pmatrix}$, where 100% of the time task τ_i has C_i worst-case execution time. Consequently, probabilistic analyses generalize the results of deterministic ones. The period in its probabilistic model can be defined as $\mathcal{T}_i = \begin{pmatrix} T_{i,k} \\ \mathbb{P}(T = T_{i,k}) \end{pmatrix}_{k \in \{1,\cdots,k_{T_i}\}}$; even the offset $\mathcal{O}_i$ and the deadline $\mathcal{D}_i$ can be random variables with the possible values for the task jobs. A probabilistic task τ_i is then the tuple $(\mathcal{O}_i, \mathcal{C}_i, \mathcal{T}_i, \mathcal{D}_i)$.

Classical probabilistic real-time analysis assumes **independency** among random variables in order to apply convolutions and obtain single task response time as a random variable [2, 3]. Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are independent if they describe two events such that the outcome of one event does not have any impact on the outcome of the other. This result in a joint probability distribution $\mathbb{P}\{\mathcal{X} = x \text{ and } \mathcal{Y} = y\} = \mathbb{P}\{\mathcal{X} = x\} \cdot \mathbb{P}\{\mathcal{Y} = y\}$.

[1]Deadline included, for a general view of the framework, although purists of real-time prefer to have deterministic deadlines.

Independency is the case for variables of the same task since there is not clear dependency among, for example, periods and worst-case execution time of the same task. On the other hand, there are dependencies among tasks: the execution of a task can affect other tasks in terms of their parameters, such as their periods or their execution time. The system as well as the environment can inflate dependencies among tasks through, for example, shared resources such as memories and caches or interrupts. *Complex systems have dependencies among their elements.* Therefore, probabilistic models of these systems must include dependencies among their random variables. In order to provide accurate results, these dependencies must be taken into account in probabilistic real-time analyses. For example, in case of dependencies the convolution cannot be applied anymore so we need an alternative operator for the joint distributions.

Analysis with Dependencies. Since the objective of probabilistic real-time modeling and analysis is to reduce the pessimism, what is a "dependent" task model that can be applied? The open problem is clearly *how to model dependencies among tasks and task parameters*, so to apply the probabilistic approach to realistic real-time cases.

Bernard et al [7] have interestingly coupled the notion of copula and real-time systems as a way of bounding dependencies among tasks. The copula of random variables is defined as the joint cumulative distribution function of those variables. Unfortunately, it has not been applied so far due to the complexity in finding an accurate characterization of dependencies. To conclude, a) which is an accurate model of the task dependencies? b) Is it possible to take into account dependencies in the random variable distribution representation? c) How do we ease the applicability of copulas or alternative dependency representations?

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.

[2] J. Lopez, J. L. Daz, J. E., and D. Garca, "Stochastic analysis of real-time systems under preemptive priority-driven scheduling," *Real-time Systems*, vol. 40, no. 2, pp. 180–207, 2008.

[3] L. Cucu and E. Tovar, "A framework for response time analysis of fixed-priority tasks with stochastic inter-arrival times," *ACM SIGBED Review*, vol. 3, no. 1, 2006.

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

[5] H. Zeng, M. D. Natale, P. Giusto, and A. L. Sangiovanni-Vincentelli, "Using statistical methods to compute the probability distribution of message response time in controller area network," *IEEE Trans. Industrial Informatics*, vol. 5, no. 4, pp. 678–691, 2010.

[6] L. Santinelli and L. Cucu-Grosjean, "Towards probabilistic real-time calculus," *Special issue related to the 3rd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems(CRTS 2010) at ACM SIGBED Review*, vol. 8, no. 1, March 2011.

[7] G. Bernat, A. Burns, and M. Newby, "Probabilistic timing analysis: An approach using copulas," *Journal of Embedded Computing*, vol. 1, pp. 179–194, April 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1233760.1233763

Undoing the Task: Moving Timing Analysis back to Functional Models

Marco Di Natale
Scuola Superiore S. Anna, email: marco@sssup.it

Haibo Zeng
McGill University, email: haibo.zeng@mcgill.ca

The real-time systems community has traditionally considered tasks or jobs (from the operating system concept of thread) as the units for the analysis model. With time, more complex task models have been created to represent conditional execution (branching), precedence constraints, and an increasingly complex model of time dependencies, from the multiframe model until the most recent extended digraph model. In the past, research works have explored the benefits of breaking the task structure to enforce a different management for subsets of the task execution time (for example, by restricting preemption, or changing the execution priority). Examples are the dual priority scheduling [3] and the non-preemption sections at the end of the task [4].

In the meantime, the industrial world is moving away from the traditional manual programming to adopt model-based design. The threads (as concurrent units of execution, managed by the operating system) are in the background, and functional models, such as dataflows or networks of synchronous blocks, including extended finite state machines are the modeling entities. The task (or threads) model becomes an intermediate artifact, and the timing analysis becomes part of a synthesis problem. The problem constraints are the semantic properties of the functional model that need to be preserved, and the task model must guarantee a correct implementation that is feasible and memory effective or time-robust.

I. Functional models

Most functional models in use today are built on a synchronous reactive (SR) semantics. For simplicity, we restrict to discrete-time models. The system is a network of functional blocks b_j. Blocks can be of two types. Regular blocks process a set of (discrete time) input signals at times that are multiples of a period T_j, which is in turn an integer multiple of a system-wide *base period* T_b (the model could be extended to sporadic activations). We denote inputs of block b_j by $i_{j,p}$ ($\overline{i_j}$ as vector) and outputs by $o_{j,q}$. At all times kT_j the block reads the signal values on its inputs and computes two functions: an output update function $\overline{o_j} = f_o(\overline{i_j}, S_j)$ and a state update function $S_j^{\text{New}} = f_s(\overline{i_j}, S_j)$, where S_j (S_j^{New}) is the current (next) state of b_j. Often, the two update functions can be considered as one $\overline{o_j}$, $S_j^{\text{New}} = f_u(\overline{i_j}, S_j)$. For timing analysis, the worst execution time of the update function is estimated as γ_j.

State machine (SM) blocks can have multiple activation events $e_{j,v}$ (as shown in the right hand side of Figure 1 with two events of period 2 and 5). At any integer multiple of one of the events' periods $kT_{j,v}$, an update function is computed depending on the current state, the subset of input events that are active and the set of input values. Update functions are typically extended by allowing the execution of generic functions whenever a given event is active on a given state. When multiple events are active, an order is provided to give some events (for the given state) precedence over others (thereby guaranteeing determinism). This is typically summarized in a graph representation as in the right side of Figure 1. The figure represents an SM with two events with periods 2 and 5 and the corresponding possible activation times and actions.

In the case of a state machine block, it pays off to identify the worst-case execution time associated to each update/action for each state, event and set of input values. The structure of the state machine constrains which update/actions can occur in the worst case within a given time interval (for details refer to [2]). The procedure to calculate the request and demand bound functions for state machines is similar to those used for digraph task models. For the example of Figure 1, the worst-case sequence of actions is defined by the state graph (which transitions are possible out of which state). Also, a trivial solution consists in an implementation with a single task running at the greatest common divisor of the events periods, but different task models may be defined.

If two blocks b_i and b_j are in an input-output relationship (one of the outputs of b_i is the input of b_j, and the output of b_j depends on its input), there is a communication link between them, denoted by $b_i \rightarrow b_j$. Let $b_i(k)$ represent the k-th occurrence of block b_i (belonging to the set of time instants $\bigcup_v kT_{i,v}$ if a state machine block, or kT_i if a standard block), then a sequence of activation times $a_i(k)$ is associated to b_i. Given $t \geq 0$, we define $n_i(t)$ to be the number of times that b_i has been activated before or at t.

In case of a link $b_i \rightarrow b_j$, if $i_j(k)$ denotes the input of the k-th occurrence of b_j, then the SR semantics specify that this input is equal to the output of the last occurrence of b_i that is no later than the k-th occurrence of b_j, i.e., $i_j(k) = o_i(m)$, where $m = n_i(a_j(k))$. This implies a partial order in the execution of the block functions (different from the precedence constraints assumed in task models). The top timeline on the left of Figure 1 illustrates the execution of a pair of blocks with SR semantics. The horizontal axis represents time. The vertical arrows capture the time instants when the blocks are activated and compute their outputs from the input values. In the figure, it is $i_j(k) = o_i(m)$.

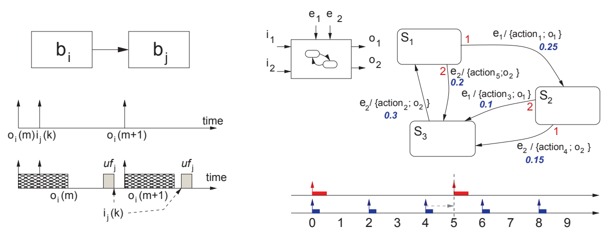


Figure 1. input-output relationship among blocks (left) / state machine and activation events (right)

The update functions and their action extensions are executed by program functions (or lines of code) executed by a task, thereby providing the level of granularity that can be leveraged to improve schedulability (for example, by disabling preemption). This makes the task structure a design artifact or objective, rather than the starting point.

The function-to-task mapping consists of a relation between a block update function (or each one of them in the case of an FSM block) and a task, and a static scheduling (execution order) of the function code inside the task. The i-th task is denoted as τ_i. $\mathcal{M}(f_{i,k}, p, n)$ indicates that the function $f_{i,k}$ of block b_i is executed as the n-th segment of code in the context of τ_p.

II. The synthesis problem

The stage of the design process in which the functional model is mapped into a task (thread) model is the starting point of several optimization problems, including how to map functions into tasks, how to assign the execution order of functions inside tasks, how to assign the task parameters (priority, deadline, offset) to guarantee semantics preservation and schedulability, how to assign scheduling attributes to functions (including preemptability and preemption threshold) and even how to design communication mechanisms that ensure flow preservation while minimizing the amount of memory used.

The bottom-left side of Figure 1 shows the possible problems with flow preservations in multi-task implementations. The writer finishes its execution producing $o_i(m)$. If the reader performs its read operation before the preemption by the next writer instance, then (correctly) $i_j(k) = o_i(m)$. Otherwise, it is preempted and a new writer instance produces $o_i(m+1)$. In case the read scheduling is delayed, the reader reads $o_i(m+1)$, in general different from $o_i(m)$.

The correct set of values may be provided to the reader by enforcing an execution order by the scheduler or by using a suitable communication mechanism (such as an instance of wait-free communication), with the associated memory overhead.

The mapping of functional blocks into tasks, the configuration of the task model, and the selection of the mechanisms for the implementation of the communication over ports (protecting against data inconsistency and possibly flow semantics violations) have a large impact on the performance of the system. The selection of the communication mechanism and the protocol to protect state variables leverages tradeoffs between time overhead for the execution of the protocol, memory required for the implementation of the mechanism, and possible blocking time. For implementation on single-CPU architecture platforms, solutions have been proposed (not exhaustively, many problems are still open).

Examples of problems that are open are the following: Given a system composed of multiple communicating state machine blocks and dataflow blocks to be executed onto a multicore platform, find the mapping of the state machine actions and block reactions onto a suitable set of tasks, the placement of such tasks onto the cores and the assignment of activation offsets and priorities to tasks such that the partial order of execution defined by the functional model semantics is preserved and each block processes its inputs and computes its next state and output in time for the next execution of the follower blocks. As starting point, a discussion on possible task implementations for a single state machine block is provided in [1] and the time analysis of finite state machine actions, implemented by a single task (with similarities to the analysis of generalized digraph models [5]) is discussed in [2].

However, other (possibly) simpler problems also exist. one example is the following: define the set of tasks that can provide an implementation to a network of block actions in a multicore platform, with the assignment of task priorities and possibly activation offsets, and the assignment of preemption thresholds to actions inside the tasks (to limit preemptability) in such a way that the implementation is correct and the use of memory (for stack and communication) is minimized.

References

[1] M. Di Natale and H. Zeng, "Task implementation of synchronous finite state machines," in *Proc. the Conference on Design, Automation, and Test in Europe*, 2012.

[2] H. Zeng and M. Di Natale, "Schedulability Analysis of Periodic Tasks Implementing Synchronous Finite State Machines," to appear in *Proc. 23rd Euromicro Conference on Real-Time Systems*, 2012.

[3] R. Davis and A. Wellings, "Dual priority scheduling," in *Proc. the 16th IEEE Real-Time Systems Symposium*, 1995.

[4] M. Bertogna, G. Buttazzo, and G. Yao, "Improving feasibility of fixed priority tasks using non-preemptive regions", in *Proc. the 32th IEEE Real-Time Systems Symposium*, 2011.

[5] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011.

The Chilling Effect of Parallelism: Analysis and Allocation of Parallel Real-Time Jobs for Peak System-Temperature Minimization

Joël Goossens[1] and Nathan Fisher[2]

[1]Université Libre de Bruxelles, Brussels, Belgium joel.goossens@ulb.ac.be
[2]Department of Computer Science, Wayne State University, Detroit, Michigan, USA, fishern@cs.wayne.edu

I. Introduction

An oft-repeated benefit of multicore platforms over computationally-equivalent single-core platforms is increased energy efficiency and thermal dissipation. For these power benefits to be fully realized, a computer system must possess the ability to parallelize its computational workload across the multiple processing cores. However, parallel computation often comes at a cost of increasing the total, overall computation that the system must perform due to communication and synchronization overhead of the cooperating parallel processes. In this document, we explore the trade-off between parallelization on real-time applications and peak-system temperature minimization.

Very little research has addressed both parallelization and power-consumption issues [8], [2] and the work that exists in this area concentrates on energy consumption and has restriction on the real-time task model [8]. While minimizing the energy consumption and peak temperature of a system are related problems, the solutions for addressing these problems are fundamentally different. For example, it is well known that in a system with dynamic voltage and frequency scaling capabilities (DVFS) the optimal frequency assignment of each processing core (in the presence of *sequential* real-time tasks) for the purpose of energy minimization is quite different than the frequency assignment for peak-temperature minimization (e.g., see Wang and Chen [12]). Clearly, since sequential tasks are a special, restrictive case of parallel tasks, this example continues to exhibit the differences in the objective for the parallel real-time task model. Unfortunately, while numerous research papers have addressed peak temperature minimization in systems comprised of non-parallelizable real-time jobs (see [11] for a survey), to the best of our knowledge, no prior research exists concerning job parallelization and peak temperature in the real-time literature. We report in this document open problems related to the scheduling of parallel real-time tasks upon multiprocessors constrained the peak temperature. We will first introduce, in sections II-A & II-B, important concepts for thermal and parallel systems, respectively.

II. Models

A. Thermal Model

We consider a multicore processing platform with a set of heat sinks for heat dissipation. Using the notation of Fisher et al. [5] in the following, the multicore processing platform $\mathcal{M}$ consists of M cores labeled by the index number, $\mathcal{M} = \{1, 2, 3, \ldots, M\}$. The thermal conductance between Cores j and ℓ in $\mathcal{M}$ is $G_{j,\ell}$, where $G_{j,\ell} = G_{\ell,j}$. We assume that the capacitance of Core j in $\mathcal{M}$ is C_j. Suppose that the thermal conductance of a core dissipating heat to the environment is $G^\dagger$. For simplicity and brevity, we do not include heat sinks in our description; however, they can easily be incorporated into the model.

The temperature of Core j is defined as $\Theta_j(t)$ and $\Theta_h(t)$. For the current problem formulation, we assume that the ambient temperature Θ_a is fixed. We also define $\Psi_j(t)$ as the power consumption on Core j at time t. As a starting point, we assume that $\Psi_j(t)$ equals αs_j^γ, where s_j is the execution speed of Core j and both γ (≤ 3) and α are processor-dependent constants. In this initial problem description, we focus only on dynamic-power consumption and ignore leakage power; however, to be realistic, a complete solution would consider both dynamic and static power. We hope that solutions for our posed open question involving dynamic power can be extended to the static-power setting.

Informally, the rate of change in the temperature on a core is proportional to the power consumption times the quantity of the heating coefficient minus the cooling coefficients times the quantity of the temperature gradients among the core and its neighboring cores. The heating/cooling process may be calculated by using the duality principle between electrical and thermal circuits and standard theory of electrical circuits:

$$C_j \frac{\mathrm{d}\Theta_j(t)}{\mathrm{d}t} = \Psi_j(t) - \sum\nolimits_{\ell \in \mathcal{M}} G_{j,\ell}(\Theta_j(t) - \Theta_\ell(t)) - G^\dagger(\Theta_j(t) - \Theta_a) \tag{1a}$$

where $\frac{\mathrm{d}\Theta_j(t)}{\mathrm{d}t}$ is the derivative of the temperature on Core j.

B. Parallel Task Models

We deal with jobs which may be executed on different processors at the very same instant, in which case we say that *job parallelism* is allowed. Various kind of task model exist, Goossens et al. [6] adapted parallel terminology [1] to recurrent (real-time) tasks as follows.

Definition 1 (Rigid, Moldable and Malleable Job): A *job* is said to be (i) *rigid* if the number of processors assigned to this job is specified externally to the scheduler a priori, and does not change throughout its execution; (ii) *moldable* if the number of processors assigned to this job is determined by the scheduler, and does not change throughout its execution; (iii) *malleable* if the number of processors assigned to this job can be changed by the scheduler during the job's execution.

At task level the literature distinguish between at least two kinds of parallelism.

- *Multithread.* Each task is sequence of phases, each phase is composed of several threads, each thread requires a single processor for execution and *can* be scheduled simultaneously [10]. A particular case is the *Fork-Join* task model where task begins as a single master thread that executes sequentially until it encounters the first fork construct, where it splits into multiple parallel threads which execute the parallelizable part of the computation [9] and so on.
- *Gang.* Each task corresponds to $e \times k$ rectangle where e is the execution time requirement and k the number of required processors with the restriction the k processors execute task in unison [7].

Assuming that a job J_ℓ has a processing requirement of e_ℓ and is assigned to k_ℓ processors for parallel execution, then several model are proposed in the literature to characterize the multiprocessor speedup vector $\Gamma_\ell = (\gamma_{\ell,1}, \dots, \gamma_{\ell,m})$ with the interpretation that job J_ℓ that executes for t time units on j processors completes $\gamma_{\ell,j} \cdot t$ units of execution.

- *sub linear speedup ratio [8]* requires that $1 \leq \frac{\gamma_{\ell,j'}}{\gamma_{\ell,j}} < \frac{j'}{j}$ where $j < j'$.
- *work-limited parallelism [3]* $\frac{j'}{j} > \frac{\gamma_{i,j'}}{\gamma_{i,j}}$ and $\gamma_{i,(j'+1)} - \gamma_{i,j'} \leq \gamma_{i,(j+1)} - \gamma_{i,j}$ where $j < j'$.
- *communication model [4]* requires that $\gamma_{\ell,j} = \frac{e_\ell}{\frac{e_\ell}{\ell} + (\ell-1)C}$ where C is the constant communication overhead cost.

III. Open Problems

Informally, our main open problem is:

> For each Core $j \in \mathcal{M}$, determine the speed/frequency assignment s_j that minimizes the peak system temperature (i.e., minimize $\max_{\ell \in \mathcal{M}}\{\max_{t>0} \Theta_\ell(t)\}$) such that all real-time parallel jobs in a recurrent task system meet their deadline.

Furthermore, we pose the above problem for each of the parallel job models (i.e., rigid/moldable/malleable).

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

SP1 *Development of Parallel Job Schedulability Analysis for Uniform Multiprocessor Platforms*. Since each core executes at a potentially different speed and any job may execute on any core, the uniform heterogeneous multiprocessor platform model is an appropriate processing abstraction. However, to the best of our knowledge no schedulability analysis exists for the parallel real-time job setting.

SP2 *Development of Thermal-Aware Online Real-Time Scheduling Algorithms*. The locality of the cores and their thermal properties are significant factors in minimizing peak temperature. Thus, a single parallel job may generate heat from multiple sources which may have some complex interaction. Therefore, an example showing the benefit of parallelism for peak-temperature minimization would be interesting and we are interested in online algorithms for determining how a parallel job should be "spread" across processors to optimize our objective.

SP3 *Development of Thermal-Aware Frequency Assignment Schemes*. An offline allocation algorithm or online scheduling algorithm is necessary to decide on what are the values of $s_j(t)$ for each $j \in \mathcal{M}$. In the offline setting, the frequency/speed allocation algorithm should determine the minimum assignment that respects deadlines and optimizes the objective function. In the online setting, the core speed could dynamically change.

References

[1] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*, chapter Scheduling Parallel Jobs on Clusters, pages 519–533. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[2] S. Cho and R. Melhem. Corollaries to Amdahl's law for energy. *Computer Architecture Letters*, 7(1):25–28, 2007.

[3] S. Collette, L. Cucu, and J. Goossens. Integrating job parallelism in real-time scheduling theory. *Information Processing Letters*, 106(5):180–187, May 2008.

[4] R. Dutton and W. Mao. Online scheduling of malleable parallel jobs. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 1–6, 2007.

[5] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling and analysis on multicore systems. *Journal of System Architecture*, 57(5):547–560, May 2011.

[6] J. Goossens and V. Berten. Gang FTP scheduling of periodic and parallel rigid real-time tasks. In *Real-Time and Network Systems*, pages 189–196, November 2010.

[7] S. Kato and Y. Ishikawa. Gang EDF scheduling of parallel task systems. In *30th IEEE Real-Time Systems Symposium*, pages 459–468. IEEE Computer Society, 2009.

[8] F. Kong, N. Guan, Q. Deng, and W. Yi. Energy-efficient scheduling for parallel real-time tasks based on level-packing. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 635–640. ACM, 2011.

[9] K. Lakshmanan, S. Kato, and R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *Real-Time Systems Symposium*, pages 259–268, December 2010.

[10] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *ECRTS*, 2012. Accepted.

[11] H. F. Sheikh, I. Ahmad, Z. Wang, and S. Ranka. An overview and classification of thermal-aware scheduling techniques for multi-core processing systems. *Sustainable Computing: Informatics and Systems*, 2011. To appear, available online.

[12] S. Wang and J.-J. Chen. Thermal-aware lifetime reliability in multicore systems. In *Proceedings of the International Symposium on Quality Electronic Design*, pages 399–405. IEEE Computer Society, 2010.

Exploiting online WCET estimates

Sverre Hendseth[1]
Department of Engineering Cybernetics, NTNU, Trondheim Norway
Email: Sverre.Hendseth@itk.ntnu.no

Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
Email: Giorgio.Buttazzo@sssup.it

***Abstract*—This problem formulation builds on the acknowledgment that, for some systems, offline WCET analysis is too conservative to be useful. For many classes of programs, better bounds on the execution time can be found effectively online by utilizing information like input data or program state. Both the systematic extension of the class of algorithms for which such bounds can be found, and the general techniques for the exploitation of the bounds are largely unexplored problems.**

I. Introduction

Having upper bounds on the execution time for the jobs in a real-time system is critically important in that guaranteeing meeting any deadline is impossible without it. The system is then, classically, built by over-allocating resources so that even in the worst case, all timing requirements are met. However, in many cases this approach is based on very pessimistic assumptions and leads to a large resource waste. Two trends have reinforced this picture in the last decades: First, the embedded systems software has become more complex both in terms of size and complexity, leading to challenges to find tight upper bounds of the execution time. Second, with the increase of battery-operated and wireless systems the overallocation of resources becomes too expensive.

This paper takes the position that offline worst-case execution time analysis is not efficient as a dimensioning factor for building and scheduling real-time systems, and we suggest exploring the better bounds that can be found online when input data and program state are available and can be used in combination with the results from the offline analysis.

The idea of utilizing online job data for improving the scheduler's decisions is not entirely novel: Mok and Chen [1] accept the potential importance for the scheduler to use more information on the execution time of a job than the worst case, but assume this data known. Choi et al. [2], take the opposite approach of letting the application itself control CPU voltage and frequency based on job parameters, but without notifying the scheduler.

In scenario-based design the connection is made between between the data available to the program and the scheduler's decision-making. Here, the possible executions of the application are partitioned, offline, into scenarios, and a prediction algorithm running online in the program's address space provides the scheduler with the information on which scenario is occurring [3].

[1]Work done at Scuola Superiore Sant'Anna, Italy

Audsley et al. [4] uses *Gain Points* in the running code to notify the scheduler of updated WCET values when the outcome of major control flow decisions has been calculated by the program. This can be e.g. just after the test of an if-statement has been performed or when a loop bound has been calculated. The application envisioned here is the execution of optional components to hard real-time programs with online guarantees.

We propose that using both offline and online information for job scheduling deserves a broader and more systematic investigation.

II. Model

A number of jobs τ_i are to be scheduled according to their deadlines D_i in presence of significant dynamicity. Each job is described by their worst-case execution time, here denoted $WCET^{off}$ and the *online WCET*, $WCET^{on}$, which will be calculated at the job's release time R_i. C_i denotes the real computation time which is unknown until the job has finished.

An estimation algorithm is assumed, which measures relevant metrics available online, and converts them to the $WCET^{on}$ by a transformation prepared off-line. The relevant input data of the job is assumed either available at the release time or inexpensive to acquire by the algorithm.

As a job becomes ready, the scheduler immediately runs the corresponding estimation algorithm and then utilizes the gained information in its decision making. The estimation algorithm is assumed to have an execution cost that, while not being insignificant, will be very small compared to that of the job itself.

The existence of effective estimation algorithms is not obvious: In the easiest case, information like the type of the frame to be processed, or the length of the list to be traversed, is directly available at job release time. On the other end of the scale we have programs for which termination itself is an open question, yielding the nonexistence of effective estimation algorithms. Between these extreme cases we have any number of program categories, potentially requiring different techniques for estimation and inviting different bound qualities.

In a real-time setting we know that we never deploy programs that do not terminate in bounded time. This can be taken to indicate that we will not be struggling with the hardest categories of programs.

$Crit$	Job	$WCET^{off}$	$WCET^{on}$	C	R	D
Hard	τ_1	12	9	8	0	16
Soft	τ_2	8	6	5	0	10

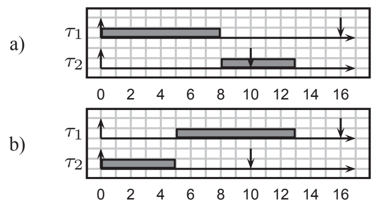


Fig. 1. Scheduling can be improved for this mixed criticality job set if $WCET^{on}$ is used, enabeling the schedule in b). Using only $WCET^{off}$ forces the schedule in a) where τ_2 misses its deadline.

III. Potential utilization areas of online WCET data.

The following points envision some possible application areas for the online WCET:

- Some schedulability or timeliness properties of a system could be refined using the $WCET^{on}$.
- Algorithms for reclaiming the CPU capacity allocated to a job but not used, might be improved by getting better information on the real job load at the job release time. Similarly, strategies for overload handling, jitter reduction, power optimization etc. could potentially be improved.
- Figure 1 shows a simple mixed criticality job set where access to the tighter $WCET^{on}$ improves schedulability.
- In a quality of service perspective, programs might offer more modes of operation. Comparing the online WCET of each mode with the available CPU for the next period might be useful for selecting the mode.
- Algorithms for dynamic deployment of jobs onto processors and for load balancing can be improved using the online execution time estimates.

IV. Principles for online estimation of WCET.

The following points suggest how the estimation challenge can be approached for increasing algorithm complexity:

- The trivial case is when the estimation metric is directly available. This can be the “message size”, “frame type”, “length of the linked list”, etc. The estimation algorithm would just look up the information and translate it into the online WCET.
- A slightly worse class of programs would be when a subset of the calculations must be carried out by the estimation algorithm - like calculating the bounds of the significant loops or the outcomes of the major conditionals. Figure 2 gives an example of a type of program where this would be successful. A framework for automatically generating such estimation algorithms is under development.

```
for (m=1;m<mmax;m+=2) {
  for (i=m;i<=n;i+=istep) {
    j=i+mmax;
    tempr=wr*data[j]-wi*data[j+1];
    tempi=wr*data[j+1]+wi*data[j];
    data[j]=data[i]-tempr;
    data[j+1]=data[i+1]-tempi;
    data[i] += tempr;
    data[i+1] += tempi;
  }
  wr=(wtemp=wr)*wpr-wi*wpi+wr;
  wi=wi*wpr+wtemp*wpi+wi;
}
```

(a)

```
g_estimate += 1;
for(m=1;m < mmax;m+=2){
  g_estimate += 13;
  for(i=m;i <= n;i+=istep){
    g_estimate += 32;
  }
}
```

(b)

Fig. 2. a) shows the two inner loops of the Numerical Recipes FFT implementation. A corresponding algorithm for estimating its the execution time has been automatically generated in b) (here plainly counting the applications of C binary operators). Further optimizations are apparent.

- Even more challenging could be a program which was heavy on control-decisions like a sorting or Huffman decoding program. However, on one hand many of these algorithms are well analyzed already so that suitable bounds may be found in the literature. On the other hand, good and inexpensive metrics might still be found by accepting uncertainties on the lower levels of granularity.

V. Open Problems

The two proposed open problems are:

- The systematic extension of the class of programs for which effective estimation algorithms exists which yields tight execution time bounds by utilizing data available online, like input data or program state.
- The corresponding utilization of these online execution time bounds in making systems with better timeliness properties.

References

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

[2] Kihwan Choi, Karthik Dantu, Wei-Chung Cheng, and Massoud Pedram, “Frame-based dynamic voltage and frequency scaling for a mpeg decoder,” in *IN ICCAD 2000*, 2002, pp. 732–737.

[3] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. De Bosschere, “System scenario based design of dynamic embedded systems,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, January 2009.

[4] N.C. Audsley, R.I. Davis, and A. Burns, “Mechanisms for enhancing the flexibility and utility of hard real-time systems,” in *Real-Time Systems Symposium, 1994., Proceedings.*, dec 1994, pp. 12 –21.

Can Randomness Buy Clairvoyance? A Look into Stochastic Scheduling of Mixed Criticality Real-Time Job Systems with Execution Time Distributions

Bader Alahmad   Sathish Gopalakrishnan
University of British Columbia, Vancouver, BC, Canada

I. Introduction

Consider a system where n (one-shot) jobs contend for the time of a single machine. In an L-criticality levels system, a job, instead of being associated with a single nominal Worst Case Execution Time Estimate (WCET), is rather associated with WCET estimates at L "degrees" of overload. A higher overload degree is quantified by higher resource demands (e.g., greater WCET). The system operates at criticality level greater than one if at least one job experiences an overload condition and thus executes beyond its nominal WCET. A job's criticality codifies the condition(s) under which a job's execution has infinite marginal utility over jobs with lower criticalities. In other words, jobs whose criticality is less than the system criticality level can be dropped altogether without compromising the safety and performance of the system, in order to allow higher criticality jobs to finish their execution fruitfully.

In the deterministic setting there is no clue as to which criticality level the input job system is more likely to operate. In this case, the system's future behavior is purely adversarial. In contrast, when the probability distributions of job execution times at different criticality levels are known, we can take a (possibly hazy) glimpse into the future evolution of the system execution behavior. That is, such distributional information will admit predictions about the behavior that the system is most likely to exhibit, conditioned on the past. Those predictions can be used to guide execution time allocation, in order to reduce the pessimism. Further, they will assist in deciding the earliest times at which to drop those jobs whose criticalities are less than that of the realized system behavior.

II. Review of Deterministic Mixed-Criticality Scheduling (Baruah et al. [1], [2])

A. System Model

The system is composed of n independent jobs, denoted as $\mathcal{J} = \{J_1, \ldots, J_n\}$, to be scheduled preemptively on a single machine, with $L \in \mathbb{N}$ criticality levels. Every job $J_i \in \mathcal{J}$ is associated with a release time $r_i \in \mathbb{Q}_+$, an absolute deadline $d_i \in \mathbb{Q}_+$, and a fixed criticality $\chi_i \in \{1, \ldots, L\}$. A greater integer value corresponds to a higher criticality. At each criticality level $\ell \in \{1, \ldots, L\}$, every job is associated with a WCET estimate $c_{i,\ell} \in \mathbb{Q}_+$. Accordingly every job $J_i \in \mathcal{J}$ is completely characterized as $J_i = (r_i, d_i, \chi_i, \langle c_{i,1}, \ldots, c_{i,L} \rangle)$.

B. Behavior of Job Execution

The actual execution time that every job consume as it executes upon the processor at run-time cannot be determined a priori; rather this information is revealed only when every job signals that it has finished execution.

A realization of job execution times for all jobs is called an *execution behavior*[1]. A behavior of execution of a job set, $\mathcal{J}$, is defined as $\mathbf{b} = \langle b_1, \ldots, b_n \rangle$, where $b_i \in \mathbb{Q}_+$. Job behaviors are therefore the *uncertainties* that any scheduling algorithm needs to combat in order to achieve the maximum useful utilization of the processor. In general, $\mathcal{J}$ might exhibit infinitely many behaviors, unless we assume that jobs always execute at their WCET estimates, in which case there can be at most L^n behaviors. We shall assume the following

1) Every job's WCET estimates monotonically increase with respect to criticality levels: $c_{i,\ell} \leqslant c_{i,\ell+1}$ for every $\ell \in \{1, \ldots, L-1\}$, and
2) a job's WCET estimate does not increase beyond its own criticality: For every $J_i \in \mathcal{J}$, $c_{i,\ell} = c_{i,\chi_i}$ for every $\ell \in \{\chi_i + 1, \ldots, L\}$.

Relative to every possible behavior of $\mathcal{J}$, the *system criticality level* $\texttt{crit}_{\mathcal{J}}(\mathbf{b})$ is defined as the minimum $\ell \in \{1, \ldots, L\}$ such that every $c_{i,\ell}$ upper bounds its corresponding observed behavior component b_i as tightly as possible. Formally,

$$\texttt{crit}_{\mathcal{J}}(\mathbf{b}) = \min\left\{\ell \in \{1, \ldots, L\} : b_i \leqslant c_{i,\ell},\ \forall i \in \{1, \ldots, n\}\right\},$$

and if no such ℓ exists, then $\mathbf{b}$ is said to be *erroneous*.

C. Problem Statement

In the original problem definition ([1], [2]) it is assumed that if the system criticality level is revealed, then all jobs whose criticality is less than the system criticality level need not be considered for schedulability and can be dropped in the analysis altogether. Combining the definitions and assumptions stated above leads to the definition below of the Mixed-Criticality (MC)-Schedulability problem.

Definition 1 (MC-Schedulability)**.** *A job system $\mathcal{J}$ is MC-Schedulable if for every valid (non-erroneous) behavior $\mathbf{b}$, every job $J_i \in \mathcal{J}$ with $\chi_i \geqslant \texttt{crit}(\mathbf{b})$ can receive b_i units of execution during $[r_i, d_i]$.*

Baruah et al. [1] showed that MC-Schedulability is NP-Hard in the strong sense, even when all jobs arrive at time 0. An exception is the case where all jobs have a common deadline, where MC-Schedulability becomes polynomially decidable. MC-Schedulability with arbitrary job deadlines is still not known to reside in the class NP, and hence the question of whether or not MC-Schedulability is NP complete is open.

Next we make precise what a scheduling strtategy is.

Definition 2 (Scheduling Policy)**.** A scheduling policy *is a partial function $S : \mathbb{Z}_+ \to \mathcal{J}$ that determines, either deterministically or randomly, the job to be assigned the processor at every time instance.*

[1]Some authors use the term "scenario" synonymously to "behavior".

The most general statement of the *MC-Scheduling problem* is: Design an *on-line non-clairvoyant scheduling policy* that, given instance $I = (\mathcal{J}, L)$, achieves the goal stated in Definition 1.

III. Probabilistic Mixed-Criticality: Probabilistic Job Criticality Levels

We propose a model in which there is a probability that a job executes at a certain criticality level, and therefore a probability that a job hits a certain WCET estimate.

We assume that we are given as input the distributions of job execution times at all criticality levels as follows. Define the discrete random variable $\mathcal{C}_i : C_i \to C_i$ as $\mathcal{C}_i(c) = 1_{C_i}(c)c$. The probability mass function (pmf) of $\mathcal{C}_i$, $f_{\mathcal{C}_i}(c) \equiv \mathbb{P}(\mathcal{C}_i = c)$, reads

$$f_{\mathcal{C}_i} = \begin{pmatrix} c_{i,1} & \dots & c_{i,\chi_i} \\ f_{\mathcal{C}_i}(c_{i,1}) & \dots & f_{\mathcal{C}_i}(c_{i,\chi_i}) \end{pmatrix},$$

where $\sum_{\ell=1}^{\chi_i} f_{\mathcal{C}_i}(c_{i,\ell}) = 1$, and $f_{\mathcal{C}_i}(c_{i,\ell})$ being the probability that J_i consumes $c_{i,\ell}$ units of execution (and thus that J_i operates at criticality level ℓ), assuming that jobs always execute only at their WCET estimates. Thus every job is completely characterized as $J_i = (r_i, d_i, \chi_i, f_{\mathcal{C}_i}), i \in \{1, \dots, n\}$. We will assume that the random variables $\{\mathcal{C}_i\}_{i=1}^n$ are *mutually independent*, because we do not consider resource sharing in our model.

A. Probabilistic Model and Problem Statement

Since all job parameters are non-negative rational numbers, we do not lose generality when working with non-negative integer parameters; we can move from one number system to another by a simple scaling procedure that depends on the magnitudes of the numbers in the given problem instance.

We model the system criticality level as it evolves in time as a discrete-time stochastic process. Suppose that a randomized scheduling algorithm, say S, schedules the input job system as follows: at each epoch t, starting at $t = 0$, S picks (according to some rule that depends on job execution time distributions) one of the jobs that are ready to execute at time t, and allows it to execute upon the processor for one time unit. We associate with every $J_i \in \mathcal{J}$ a discrete-time stochastic process $\mathcal{X}^{(i)} = \{\mathcal{X}_t^{(i)}\}_{t \in \mathbb{Z}_+}$, where $\mathcal{X}_t^{(i)}$ is an indicator random variable that we define as:

$$\mathcal{X}_t^{(i)} = \begin{cases} 1 & \text{if algorithm } S \text{ assigns the processor to } J_i \text{ at } t, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{B}_m^{(i)} = \sum_{t=0}^m \mathcal{X}_t^{(i)}$ be the total execution time allocated to job J_i at $t = m$, starting from $t = 0$, by algorithm S. We define the stochastic process $\mathcal{K} = \{\mathcal{K}_m\}_{m \in \mathbb{Z}_+}$ to capture the evolution of the system criticality level according to the (probabilistic) execution-time allocation of algorithm S, where

$$\mathcal{K}_m = \min\left\{\ell \in \{1, \dots, L\} : \mathcal{B}_m^{(i)} \leqslant c_{i,\ell} \quad \forall i \in \{1, \dots, n\}\right\}.$$

Every decision made at every time instance causes a subset of the random variables comprising our stochastic processes to be realized (instantiated). Those instantiations are exactly the information that the decision procedure is allowed to use and condition upon to steer future decisions. We express this history information abstractly as "raw data", in terms of sigma-algebras, as follows. Let $\mathcal{F}_m^{\mathcal{X}^{(i)}} \equiv \sigma(\mathcal{X}_0^{(i)}, \dots, \mathcal{X}_m^{(i)})$ be the smallest σ-field that contains $\bigcup_{t=0}^m \sigma(\mathcal{X}_t^{(i)})$. Further, let $\mathcal{F}^{\mathcal{X}^{(i)}} = \{\mathcal{F}_m^{\mathcal{X}^{(i)}}\}_{m \in \mathbb{Z}_+}$ be the natural filtration of $\mathcal{X}^{(i)}$, where $\mathcal{F}_m^{\mathcal{X}^{(i)}} \subset \mathcal{F}_{m+1}^{\mathcal{X}^{(i)}}$ for every $m \in \mathbb{Z}_+$. Then the process $\{\mathcal{B}_m^{(i)}\}_{m \in \mathbb{Z}_+}$ is $(\mathcal{F}^{\mathcal{X}^{(i)}})$-adapted; that is, $\mathcal{B}_m^{(i)}$ is $\mathcal{F}_m^{\mathcal{X}^{(i)}}$-measurable for every m. Let $\mathcal{G}_m \equiv \sigma(\mathcal{B}_m^{(1)}, \dots, \mathcal{B}_m^{(n)})$ be the smallest σ-field that contains $\bigcup_{i=1}^n \sigma(\mathcal{B}_m^{(i)})$. Then the process $\mathcal{K}$ is $(\mathcal{G}_m)$-adapted.

Let $\mathcal{T}_i$ be a random time defined as $\mathcal{T}_i = \sup\{t \geqslant r_i : \mathcal{X}_t^{(i)} = 1\}$. $\mathcal{T}_i$ is the last time that job J_i is assigned the processor by algorithm S. Thus $\mathcal{T}_i$ is either the finish time of job J_i, which occurs when J_i signals that it has executed fully, or the time when algorithm S has decided to drop J_i from the schedule. Since job execution times are finite, it follows that $\mathcal{T}_i < \infty$ almost surely. In particular, the *makespan* of any work conserving[2] schedule of the input job system, which is the completion (finish) time of the last job to leave the system, is upper bounded by $\widetilde{M} = \max_i\{r_i\} + \sum_{i=1}^n c_{i,L}$, independently of the underlying scheduling strategy. Therefore, $\mathcal{T}_i \leqslant \widetilde{M}$ almost surely, and $\{\mathcal{T}_i = m\} \equiv \{\mathcal{X}_m^{(i)} = 1, \mathcal{X}_{m+1}^{(i)} = 0, \dots, \mathcal{X}_{\widetilde{M}}^{(i)} = 0\}$. We may note that the system criticality level can only either increase or stay at the same value, thus $m \mapsto \mathcal{K}_m$ is monotonically increasing, and $(\mathcal{K}_{m+1} - \mathcal{K}_m) \in \{0, 1\}$ almost surely.

Define the random time $\mathcal{T} = \max_i \mathcal{T}_i$. Intuitively, $\mathcal{T}$ is the time at which the system criticality level is revealed, and thus is the makespan of the job schedule according to algorithm S (if $r_i = 0\ \forall J_i \in \mathcal{J}$, then equivalently $\mathcal{T} = \sum_{i=1}^n \mathcal{B}_{\mathcal{T}_i}^{(i)}$).

Open Problem: *Stochastic Mixed Criticality Scheduling* (`MC-STOCH`)
We are concerned with designing a scheduling policy that makes probabilistic decisions such that, on a given instance $I = (\mathcal{J}, L)$

(i) for every job $J_i \in \mathcal{J}$, when the realized system criticality level is greater than job J_i's criticality, then J_i receives as little execution time as possible with high probability; that is, $\mathbb{P}\left(\mathcal{B}_{\mathcal{T}}^{(i)} = 0, \mathcal{K}_{\mathcal{T}} > \chi_i\right)$ is high for every $J_i \in \mathcal{J}$;
(ii) when the realized system criticality level is ℓ, then all jobs whose criticality is greater than or equal to ℓ receive as much execution time as their WCET at criticality level ℓ with high probability; that is, $\mathbb{P}\left(\mathcal{B}_{\mathcal{T}}^{(i)} \in (c_{i,\ell-1}, c_{i,\ell}], \mathcal{K}_{\mathcal{T}} = \ell\right)$ is high for every $J_i \in \mathcal{J}$ with $\ell \leqslant \chi_i$, and
(iii) jobs whose criticality is greater than the realized system criticality level finish before their deadlines with high probability; that is, $\mathbb{P}(\mathcal{T}_i \leqslant d_i, \mathcal{K}_{\mathcal{T}} = \ell)$ is high for every $J_i \in \mathcal{J}$ with $\chi_i \geqslant \ell$.

References

[1] S. K. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010.

[2] ——, "Mixed-criticality scheduling: Improved resource-augmentation results," in *CATA*, 2010.

[2]A work conserving scheduling strategy is one that does not purposely idles a job when the processor is not busy and a job is available for processing.

Is it possible to schedule target sensitive recurring tasks for increased utility accrual using task level rather than job level information?

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

I. Preliminaries

Real-time systems are traditionally defined as computing systems that must react to events within precise time constraints in order to provide correct behavior [2]. Therefore, most real-time schedulers were developed having the deadline as a primary concern. Deadline-based timeliness criteria are used to express an interval of time (*execution window*) where the task is allowed to execute. As a result, there is the assumption that the utility of tasks is constant within their execution window [9].

Some applications have *target sensitive constraints*: each task should preferably execute at a specific target point within its execution window, called *target point*, but can execute around this point, albeit at lower utility. Ideally, all executions would be scheduled directly at the respective target points, but it might not be feasible due to overlapping executions. Under this condition, the execution of tasks must be scheduled so that no timing constraints are violated and the accrued system utility is maximized.

Time utility function (TUF) scheduling as presented in [3], [12], [9] and earliness/tardiness schedulers [1] go beyond the starttime-deadline notion to express tasks' temporal constraints. In TUF schedulers, tasks aggregate a given amount of utility to the system as a function of when they execute; the goal of the scheduler is to maximize system utility. A study on several types of time utility functions is presented in [8], but no solution is presented. A best-effort solution for resource allocation in computing systems is proposed in [11]. However, scheduling decisions are based only on the utility that tasks accrue to the system at the current point in time and not on the shape of their utility functions over time. A TUF scheduler assuming any kind of utility function is proposed in [3]. It uses a heuristic to find an ordering on average close to the optimum in $O(N^3)$, N being the number of jobs being scheduled, under the constraint that all jobs are within the same busy period. In [12], the same problem is solved in $O(N^2)$ assuming only non-increasing TUFs. The result is used in ethernet packet scheduling to define the ordering. Not the sending times are considered, but the utility decreases upon delayed arrival. Therefore, packets are sent as early as possible after being ordered. This work is extended in [13] and [10] to support variable cost functions and mutual exclusion of resources, respectively. In [14] an energy-aware TUF scheduler is proposed, but the focus is to satisfy statistical performance requirements.

In [4], [5], we proposed the gravitational task model for target sensitive applications which allows each job to express target point and utility function. This task model is based on a physical pendulum analogy which makes understanding of the problem and solution intuitive. We presented a method with linear complexity to compute the trade-off among the execution of jobs for improved accrued utility. This method is based on an analogy with the *equilibrium* of pendulum systems, and approximates the optimum under the assumption that jobs have the same type of utility function—elliptical. In [6], we presented an on-line scheduler with complexity $O(N \times log(N))$ for the gravitational task model that uses the equilibrium, and a heuristic to reorder the execution sequence of jobs, which also impacts on the utility accrual of the schedule. These work on the gravitational task model have the limitation that jobs within a hyper-period must be scheduled at the beginning of the hyper-period, which incurs high overhead at runtime. This limitation was overcome in [7], where we proposed an EDF-based scheduling algorithm for the gravitational task model that reduces the maximum number of jobs in any equilibrium calculation at runtime to n^2, n being the number of periodic tasks in the task set.

II. How to exploit the knowledge about task periodicity to reduce scheduling complexity?

Let us consider the scheduling of recurring tasks τ_i, where $\tau_{i,j}$ represents the j^{th} job (or instance) of task τ_i. TUF schedulers and the gravitational task model consider the utility $f_{i,j}(t)$ that each job accrues when executing at t to take scheduling decisions. The goal of the scheduler is to schedule the execution of each job such that all earliest start time and deadline constraints of each job are met and to maximize equation 1. However, having a complete description of every single job that will run in the system is impractical for recurring tasks. The number of their instances depend on how long the system will run, and even the number of jobs within a repetitive scheduling cycle may have a factorial number of jobs with respect to the number of recurring tasks.

$$\text{maximize: } \sum_{i=1,j=1}^{i=n,j=num.\ instances\ \tau_i} f_{i,j}(t), \tag{1}$$

where n is the number of recurring tasks.

The question is how to schedule recurring target sensitive tasks at a task level rather than a job level. In other words, the scheduling algorithm should take decisions based on parameters of every task τ_i, whose amount is limited and independent of the lifetime of the system. Is it possible to design a scheduler capable of guaranteeing timing constraints and increasing the utility accrual without going down to the job level information of every single job $\tau_{i,j}$? Using the parameters of active jobs (i.e. jobs ready for execution), as for example done in EDF, is also acceptable, provided that the absolute parameters of each job $\tau_{i,j}$ can be derived from relative parameters of their tasks τ_i.

We believe there might be a way to provide timing guarantees and take scheduling decisions which are utility-aware and require only task level and/or active job information, but the impact on the scheduling overhead and utility accrual is unclear. At task level, each task τ_i would have a utility function $f_i(?)$ and the scheduler's goal is to maximize equation 2. The parameter over which we would define this function is unknown at the moment, which raises the question "what is the meaning of taking scheduling decisions based on the utility functions of tasks from the scheduling and the applications' perspective?".

$$\text{maximize: } \sum_{i=1,j=1}^{i=n,j=num.\ instances\ \tau_i} f_{i,j}(t), \tag{2}$$

where n is the number of recurring tasks.

Furthermore, deferring work so that tasks can meet their target sensitive constraints may compromise the timing guarantees of subsequent jobs, hence making the scheduling problem more difficult. The gravitational task model has the advantage of being work deferrable over the TUF schedulers mentioned in the previous section, a very important feature for attaining higher utility accrual [7]. Therefore, we would like to discuss on how to extend or modify the equilibrium calculation to exploit the knowledge about task periodicity to reduce the scheduling complexity. We are also open for alternative solutions or insights, if any.

III. Summary

In this paper, we discuss an open problem on the scheduling of recurring target sensitive real-time tasks. Each of these tasks has the constraint that execution is preferable at a specific target point within its execution window, called *target point*, but can execute around this point for the sake of feasibility, albeit at lower utility. To the best of our knowledge, all utility-aware scheduling algorithms available in the literature take scheduling decisions based on the utility function of each individual jobs that executes in the system, whether an instance of a recurring task or not.

We question the need to consider the utility information at job level, which incurs high scheduling overhead, and invite the attendees of the workshop to discuss on the feasibility of taking scheduling decisions based on the utility function at a task level. We are particularly interested in the gravitational task model, which tailors the scheduling of target sensitive tasks, but would also like to discuss any insights on alternative solutions.

References

[1] K. Bülbül, P. Kaminsky, and C. Yano, "Preemption in single machine earliness/tardiness scheduling," *J. of Scheduling*, vol. 10, no. 4-5, pp. 271–292, 2007.

[2] G. C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Santa Clara, CA, USA: Springer-Verlag TELOS, 2004.

[3] K. Chen and P. Muhlethaler, "A scheduling algorithm for tasks described by time value function," *Real-Time Syst.*, vol. 10, no. 3, 1996.

[4] R. Guerra and G. Fohler, "A gravitational task model for target sensitive real-time applications," in *ECRTS08 - 20th Euromicro Conference on Real-Time Systems*, Prague, Czech Republic, July 2008.

[5] ——, "A gravitational task model with arbitrary anchor points for target sensitive real-time applications," *Real-Time Syst.*, vol. 43, no. 1, 2009.

[6] ——, "On-line scheduling algorithm for the gravitational task model," in *ECRTS09 - 21th Euromicro Conference on Real-Time Systems*, Dublin, Ireland, July 2009.

[7] ——, "On-line scheduling of target sensitive periodic tasks with the gravitational task model," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 578 –581.

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

[9] P. Li, B. Ravindran, and E. D. Jensen, "Adaptive time-critical resource management using time/utility functions: Past, present, and future," in *COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts - (COMPSAC'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 12–13.

[10] P. Li, H. Wu, S. B. Ravindran, and E. D. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Trans. Comput.*, vol. 55, no. 4, 2006.

[11] C. D. Locke, "Best-effort decision-making for real-time scheduling," Ph.D. dissertation, Pittsburgh, PA, USA, 1986.

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

[13] H. Wu, U. Balli, B. Ravindran, and E. D. Jensen, "Utility accrual real-time scheduling under variable cost functions," in *Proceedings of RTCSA'05*. Washington, DC, USA: IEEE Computer Society, 2005.

[14] H. Wu, B. Ravindran, E. D. Jensen, and P. Li, "Energy-efficient, utility accrual scheduling under resource constraints for mobile embedded systems," *Trans. on Embedded Computing Sys.*, vol. 5, no. 3, 2006.

Towards optimal priority assignments for real-time tasks with probabilistic arrivals and execution times

Dorin Maxim
INRIA Nancy Grand Est
615 rue du Jardin Botanique
54600 Villers les Nancy
dorin.maxim@inria.fr

***Abstract*—In this paper we present the problem of optimal priority assignments in fixed priority preemptive single processor systems where tasks have probabilistic arrivals and execution times. We show that Rate Monotic is not optimal for our problem.**

I. Introduction

In embedded real-time systems there is a strong demand for new functionality that can only be met by using advanced high performance microprocessors. Building real-time systems with reliable timing behaviour on such platforms represents a considerable challenge. Deterministic analysis for these platforms may lead to significant overprovision in the system architecture, effectively placing an unnecessarily low limit on the amount of new functionality that can be included in a given system. An alternative approach is to use probabilistic analysis. Probabilistic analysis techniques rather than attempting to provide an absolute guarantee of meeting the deadlines, provide the probability of meeting the deadlines.

II. Model and notations

In this paper, we consider a task set of n synchronous tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$. Each task τ_i is characterized by three parameters $(\mathcal{C}_i, \mathcal{T}_i, \mathcal{D}_i)$ where $\mathcal{T}_i$ is the inter-arrival time (commonly known as period), $\mathcal{D}_i$ the relative deadline, and $\mathcal{C}_i$ the worst-case execution time. The parameters are described by random variables[1].

In the state of the art there are analysis frameworks and scheduling algorithms for tasks task-sets that have only on parameter given as a random variable, either just the execution time [1], [2], either just the period [3]. The model that we are taking into account is an extension of the these ones, and, to our knowledge, there are no existing solutions for task systems that have more than one parameter given as random variables.

A random variable $\mathcal{X}_i$ describing a parameter of τ_i is assumed to have a known probability function (PF) $f_{\mathcal{X}_i}(\cdot)$ with $f_{\mathcal{X}_i}(x) = P(\mathcal{X}_i = x)$ giving the probability that τ_i has the mentioned parameter equal to x. The values of $\mathcal{X}_i$ are assumed to belong to the interval $[x_i^{\min}, x_i^{\max}]$.

[1]In this paper we will use a calligraphic typeface to denote random variables.

For instance the worst-case execution time $\mathcal{C}_i$ can be written as follows:

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

where $\sum_{j=0}^{k_i} f_{\mathcal{C}_i}(C_i^j) = 1$.

For example for a task τ_i we might have a worst-case execution time $\mathcal{C}_i = \begin{pmatrix} 2 & 3 & 25 \\ 0.5 & 0.45 & 0.05 \end{pmatrix}$; thus $f_{C_i}(2) = 0.5$, $f_{C_i}(3) = 0.45$ and $f_{C_i}(25) = 0.05$.

All jobs are assumed to be independent of other jobs of the same task and those of other tasks, hence the execution time of a job does not depend on, and is not correlated with, the execution time of any previous job.

For example, given a tas-kset $\tau = \{\tau_1, \tau_2\}$ composed of two tasks, where $\tau_1 = \left(\begin{pmatrix} 3 & 4 \\ 0.1 & 0.9 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0.7 & 0.3 \end{pmatrix}, \begin{pmatrix} 3 & 4 \\ 0.1 & 0.9 \end{pmatrix} \right)$, and $\tau_2 = \left(\begin{pmatrix} 6 & 8 \\ 0.2 & 0.8 \end{pmatrix}, \begin{pmatrix} 2 & 4 \\ 0.6 & 0.4 \end{pmatrix}, \begin{pmatrix} 6 & 8 \\ 0.2 & 0.8 \end{pmatrix} \right)$, the first random variable of the task representing its probabilistic execution time and the second one, its release distribution.

Let us presume that τ_1 has the higher priority and τ_2 the lower priority. In this case there are multiple scenarios that can occur. We present in Figure 1 and in Figure 2 two of this possible scenarios. In the first one, τ_1 has four jobs represented andτ_2 has two jobs, the first one having an execution time equal to 2 and being released at $t = 0$ and the second job having an execution time of 4 and being released at $t = 6$. Both of these job finish execution before their respective deadlines. In the second scenario, there are represented two jobs of τ_1 and only one job of τ_2, but in this case the job of tau_2 misses its deadline. The difference now is that $\tau_{2,1}$ has an execution time equal to 4 which makes it miss its deadline at $t = 6$.

There are multiple scenarios like this and, in consequence, many questions that arise. We present some of these questions in the next section.

III. Open problems

One of the first questions that comes to mind in a probabilistic time system is *how does one analytically compute the response time distributions of the different jobs of given tasks?*

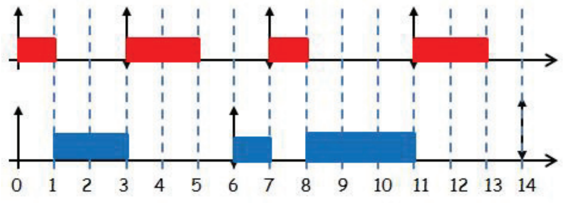


Fig. 1. Scenario 1

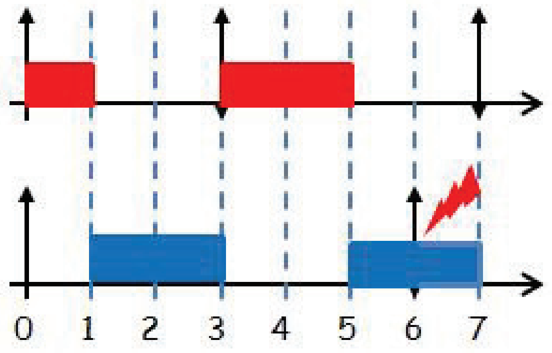


Fig. 2. Scenario 2

This is needed in order to compute the probability that the job misses its deadline, and to know the percentages of deadline misses.

Presuming that such an analytical tool is at hand, the next question that arises, and maybe the most important one in what concerns the design of a real time system is *how should the tasks be scheduled so that each task meets certain conditions referring to its timing failures?* We mention that we are searching for a fixed priority scheduling, in a preemptive context, i.e., all jobs of the same task have the same priority. The timing, or deadline failure, is usually given as a maximum percentage of deadlines that the task can miss in certain time interval. We refer to a priority ordering that meets such a requirement as a feasible priority assignment. We note that this use of the term feasible is an extension of its normal use in the deterministic case, where a feasible priority ordering is one in which the associated schedule has zero probability of timing failure.

The next question that immediately comes to mind is *how does one define a study interval in such a system?* In the deterministic case the study interval is, usually, the hyper-period. Knowing that everything that happens during a hyper-period will repeat for the next ones, it is enough to study one hyper-period to obtain the behavior of the entire system. In the probabilistic case the hyper-period might not be the answer to the question, since what happens in one hyper-period might be completely different than what happened in the previous one and what will happen in the next one. Also, there is the question of how does one compute the hyper-period in a probabilistic system.

In the following we talk about the case of the Rate Monotonic priority assignment algorithm and its compatibility with a probabilistic task-system.

A. Non-optimality of Rate Monotonic

We know from [2] that Rate Monotonic is not optimal for the problem of scheduling tasks according to a fixed-priority policy in the case of tasks with deterministic arrivals and probabilistic executions times.

Following the reasoning applied in [2], it is easy to prove that Rate Monotonic does not provide a feasible scheduling, since it does not take into account the probabilistic character of the tasks. Furthermore, in the case of tasks with probabilistic arrivals it would be difficult to determine the ordering of the tasks since each task may have multiple values representing its arrival time and even if one would apply a convention as considering the minimum value of each arrival time distribution would still not provide a feasible scheduling. For example, considering a task-set $\tau = \{\tau_1, \tau_2\}$ with τ_1 defined by $(\begin{pmatrix} 4 & 10 \\ 0.01 & 0.99 \end{pmatrix}, \begin{pmatrix} 2 & 3 \\ 0.5 & 0.5 \end{pmatrix}, \begin{pmatrix} 4 & 10 \\ 0.01 & 0.99 \end{pmatrix})$ and τ_2 defined by $(\begin{pmatrix} 8 & 9 \\ 0.6 & 0.4 \end{pmatrix}, \begin{pmatrix} 2 & 3 \\ 0.5 & 0.5 \end{pmatrix}, \begin{pmatrix} 8 & 9 \\ 0.6 & 0.4 \end{pmatrix})$, one can easily see that, even if τ_1 has its smallest arrival time smaller than those of τ_2, it very rarely arrives with a period equal to 4, most of the times it has a distance of 10 units of time between instances, which Rate Monotonic does not take into consideration.

Furthermore, since the problem of scheduling tasks according to a fixed-priority policy in the case of tasks with deterministic arrivals and probabilistic executions times is a sub-problem of our problem, we can conclude that Rate Monotonic is not optimal in the case when the arrivals are probabilistic either.

References

[1] Díaz, J.L and Garcia, D.F. and Kim,K. and Lee, C.G. and Bello, L.L. and López J.M. and Mirabella, O., Stochastic Analysis of Periodic Real-Time Systems, 23rd IEEE Real-Time Systems Symposium (RTSS02), 2002, 289-300.

[2] D. Maxim and O. Buffet and L. Santinelli and L. Cucu-Grosjean and R. Davis, On the Optimality of Priority Assignment for Probabilistic Real-Time Systems, the 19th International Conference on Real-Time and Network Systems, 2011.

[3] L. Cucu and E. Tovar, A Framework for Response Time Analysis of Fixed-Priority Tasks with Stochastic Inter-arrival Times, ACM SIGBED Review, 3(1), January 2006.

Exploiting Uni-Processor Schedulabilty Analysis for Partitioned Task Allocation on Multi-Processors with Precedence Constraints

Mario Bambagini, Giorgio Buttazzo
{mario.bambagini, giorgio.buttazzo}@sssup.it
Scuola Superiore Sant'Anna
Pisa, Italy

Sverre Hendseth
sverre.hendseth@itk.ntnu.no
Norwegian University of Science and Technology
Trondheim, Norway

***Abstract*—This paper considers the problem of scheduling real-time tasks with precedence and communication constraints on heterogeneous multiprocessor systems. Most partitioned approaches statically schedule the task set by computing start times and finishing times for each task in such a way that a desired cost function is minimized. The resulting optimization problem is however highly complex. The open problem proposed in this paper is to reduce the overall complexity by transforming precedence relations into real-time constraints and exploit uniprocessor scheduling results to guarantee the task set.**

I. Introduction

The problem of task allocation and scheduling in multiprocessor systems under precedence and real-time constraints is known to be NP-Hard and has been investigated for many years.

Such a problem has become dominant with the development of Multi-Processors System-on-Chips (MPSoC), distributed embedded systems, and computer clusters. In spite of the different contexts, the common goal is to provide algorithms for the automatic allocation of tasks to optimize the computational resources.

Many algorithms [1] have been proposed in the literature and they can be divided into global and partitioned approaches. Global algorithms pick the highest priority task from a single ready queue (shared by the cores) and allocate it on an available processor. Partitioned approaches first allocate the task on the processors and then schedule them using a local scheduler. A wide range of algorithms exist, which spread from complete searches [2], [3], meta-heuristics [4], and heuristics [5].

Most of these approaches start from a task set with precedence and time constraints and produce a static schedule, stored in a table and executed in a time-triggered fashion. The approach considered in this paper proposes to transform precedence relations into activation times and deadlines for each tasks and use an online scheduling algorithm to execute them. The advantage of this approach is to exploit existing uniprocessor results for analyzing the schedulability of the task set allocated on each processor, thus reducing the complexity to find a feasible solution.

II. Model

We consider a set Φ of m heterogeneous processors and a set Γ of n preemptive real-time tasks, characterized by a set of precedence constraints.

Each processor ϕ_j has a specific type pt_j which collects processors in groups of performance. The processors are assumed to be linked together through a fully-connected network which consists of a dedicated communication link for each node pair. These links are assumed to be full-duplex and heterogeneous, meaning that data transfer may take different time for the same message size, depending on where tasks are allocated.

A task τ_i allocated on processor ϕ_j is characterized by a worst-case computation time $C_{i,j}$. Computation times of τ_i are equal for the same processor type. Precedence dependencies are represented by a direct acyclic graph. More precisely, the notation $\tau_i \rightarrow \tau_j$ indicates that τ_j has to start not earlier than τ_i finishing time plus the communication time required for data exchange. The data transmission delay is computed as the amount of data exchanged between τ_i and τ_j divided by the bandwidth guaranteed between the two hosting processors. The whole application is considered to be periodic with a period P and a relative deadline D. For the sake of simplicity, P and D are assumed to be equal. We assume that in each processor tasks are scheduled by Earliest Deadline First (EDF) [6].

III. Open Question

Most of the proposed partitioned algorithms produce a static allocation and then a static schedule on each processor, and the application feasibility is guaranteed if and only if the latest finishing time is less than or equal to the application deadline.

Our goal is to split the multi-processor scheduling problem into m different uni-processor scheduling problems and exploit the well-known theoretical results to guarantee the feasibility on each processor. In order to apply this approach, however, it is necessary to assign an activation time and a deadline to each task, so that EDF can schedule them.

For each processor, the feasibility can be checked using the Processor Demand Criterion [7] or through the offset analysis [8]. In this case the analysis is simplified because task periods

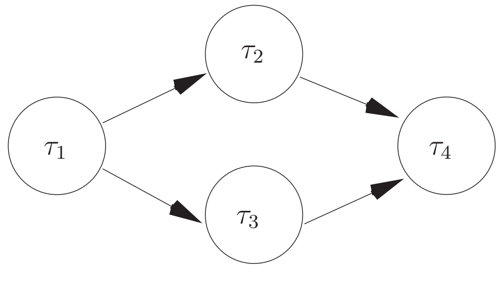


Fig. 1. Task Graph

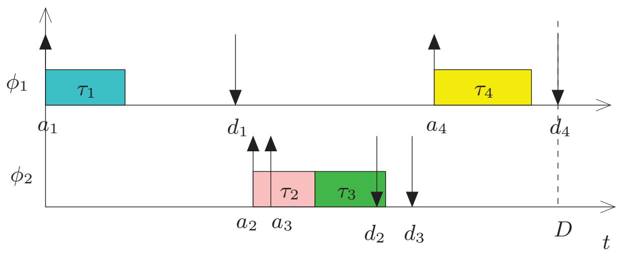


Fig. 2. A possible assignment

are all equal to the application period. This also means that the hyperperiod is equal to the application period and only one instance for each task must be taken into account within the analysis interval.

In this way, the problem mainly consists in an allocation phase and a time assignment phase, leaving the scheduling to EDF. While the allocation has been investigated for a long time, providing an effective assignment of activation times and deadlines is still an open question.

An example is reported in Figure 2 to show a possible timing constraints assignment from the precedence relations depicted in Figure 1. In the example, four tasks must be executed on two processors running EDF as a scheduler. τ_1 and τ_4 are allocated on the first processor ϕ_1 and τ_2 and τ_3 on ϕ_2. A valid timing assignment must set τ_1's deadline earlier than the activations of τ_1's successors (also considering the communication overheads). The same for τ_4.

A similar assignment problem has been addressed by Buttazzo et al. [9], who extended the idea proposed by Chetto et al. [10]. Their method, however, focuses more on the path analysis rather than on each single task and their solution assumes homogeneous systems and negligible communication costs. Moreover, for the task set illustrated in the example, assigning each flow to a different processor, their method would make τ_4's activation time occur $c_{4,1}$ units of time earlier than its deadline, meaning that τ_4 has a relative deadline equal to its computation time (Figure 3). Since their procedure would set $a_3 = a_2$ and $d_3 = d_2$, the task set would be infeasible on ϕ_2, leading τ_3 to miss its deadline. In fact, $d_3 - a_3 = d_2 - a_2 > c_{2,2} + c_{3,2}$, although a considerable amount of time is wasted till d_1.

This example suggests that it is worth investigating alternative approaches to assign activation time and deadlines that guarantee feasibility while minimizing a desired cost function.

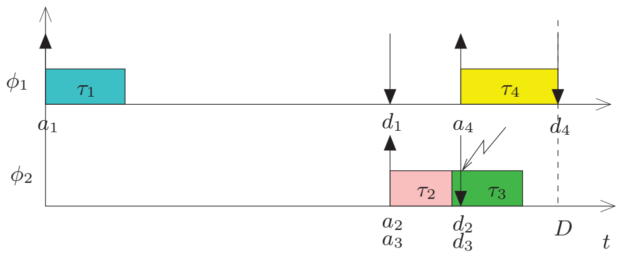


Fig. 3. Buttazzo et al.'s assignment

References

[1] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978814

[2] I. C. S. Institute, K. Shin, and D. Peng, *Static Allocation of Periodic Tasks with Precedence Constraints i Distributed Real-time Systems*, ser. Technical report (International Computer Science Institute). International Computer Science Institute, 1988. [Online]. Available: http://books.google.it/books?id=P3iIGwAACAAJ

[3] M. Lombardi and M. Milano, "Optimal methods for resource allocation and scheduling: a cross-disciplinary survey," *Constraints*, vol. 17, no. 1, pp. 51–85, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10601-011-9115-6

[4] V. V. Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, March 2010. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v201y2010i2p409-418.html

[5] L.-C. Canon, E. Jeannot, R. Sakelariou, and W. Zheng, "Comparative Evaluation of the Robustness of DAG Scheduling Heuristics," in *Integration Research in Grid Computing, CoreGRID integration workshop*, S. Gorlatch, P. Fragopoulo, and T. Priol, Eds. Hersonissos, Crete, Grèce: Crete University Press / Springer US, 2008, pp. 63–74. [Online]. Available: http://hal.inria.fr/inria-00333904

[6] C. L. Liu and J. W. Layland, "Readings in hardware/software co-design," G. De Micheli, R. Ernst, and W. Wolf, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002, ch. Scheduling algorithms for multiprogramming in a hard-real-time environment, pp. 179–194. [Online]. Available: http://dl.acm.org/citation.cfm?id=567003.567018

[7] S. K. Baruah, R. R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.

[8] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," *Real-Time Systems*, vol. 30, no. 1-2, pp. 105–128, 2005.

[9] G. C. Buttazzo, E. Bini, and Y. Wu, "Partitioning real-time applications over multicore reservations," *IEEE Trans. Industrial Informatics*, vol. 7, no. 2, pp. 302–315, 2011.

[10] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Syst.*, vol. 2, no. 3, pp. 181–194, Sep. 1990. [Online]. Available: http://dx.doi.org/10.1007/BF00365326

Energy Saving Exploiting the Limited Preemption Task Model

Mario Bambagini, Giorgio Buttazzo
{mario.bambagini, giorgio.buttazzo}@sssup.it
Scuola Superiore Sant'Anna, Pisa, Italy

Marko Bertogna
marko.bertogna@unimore.it
University of Modena and Reggio Emilia, Italy

***Abstract*—Limited preemptive scheduling has been shown to dominate both non-preemptive and fully preemptive scheduling under fixed priority systems, as far as schedulability is concerned. This paper suggests the use of DVS and DMP techniques under limited preemptive scheduling to further reduce energy consumption with respect to a fully preemptive or non-preemptive approach.**

I. Introduction

Two widely used techniques to save energy are *Dynamic Voltage and Frequency Scaling* (DVFS) and *Dynamic Power Management* (DPM). DVFS approaches decrease the voltage and/or frequency of the processor to reduce energy consumption. On the other hand, DPM techniques aim at switching the processor in a low-power inactive state as long as possible, but still guaranteeing the task real-time constraints.

Most of the approaches proposed in the last years consider a fully preemptive task model. Bini et al. [1] considered a realistic model with discrete frequencies and non-negligible overhead, and proposed to achieve the optimal speed by selecting and modulating between two available frequencies. The modulation between active and sleep state was investigated by Huang et al. [2] focusing mostly on a DPM approach. Awan and Petters [3] proposed to accumulate the task execution slack to switch the processor off during such intervals. Rowe et al. [4] presented a technique that harmonizes task periods to clusters task execution such that processor idle times are lumped together. In these papers, a limited preemptive approach was never explored to improve energy saving.

An approach to limit preemptions consists in dividing each task into a set of non-preemptive chunks by inserting a number of fixed preemption points in specific parts of the code.

As Bertogna et al. [5] have shown, limited preemptive methods increase schedulability with respect to fully preemptive and non-preemptive models, even when preemption cost is negligible. Moreover, Bril et al. [6] presented an exact schedulability analysis for fixed priority scheduling with deferred preemption.

To the best of our knowledge, only Maxim et al. [7] have addressed the problem of exploiting the limited preemption model to further reduce the energy consumption and the number of preemptions by merging the last two chunks of a task and adjusting the speed of the previous ones.

This paper suggests to combine limited preemptive scheduling with DVS and DMP techniques to further reduce energy consumption with respect to fully preemptive or non-preemptive approaches.

II. Model

We consider a set Γ of n sporadic tasks [8] $\tau_1, \tau_2, \ldots, \tau_n$ executing upon a single processor platform with preemption support. The processor can vary the running speed s, defined as the normalized frequency with respect to the nominal frequency, $s = \frac{f}{f_{nom}}$. The speed set is assumed to be finite and composed by m different speeds $s_1, s_2, \ldots, s_m$ sorted in ascending order, thus $s_{min} = s_1$, $s_{nom} = 1$ and $s_{max} = s_m$. The selected speed is set at the system start and is never changed.

Each sporadic task τ_i $(1 \leq i \leq n)$ is characterized by a worst-case execution time (WCET) $C_i(s)$, which is function of the speed, a relative deadline D_i, and a minimum inter-arrival time T_i, also referred to as the period. The WCET value of τ_i depends on the actual speed of the processor and is computed as $C_i(s) = \frac{C_i^{nom}}{s}$, where C_i^{nom} denotes the amount of required time to execute τ_i at the nominal speed (under this assumption $C_i(s_{nom}) = C_i^{nom}$). Each task generates an infinite sequence of jobs, with the first job arriving at any time and subsequent arrivals separated by at least T_i units of time.

When a task τ_i is executed with deferred preemptions, q_i^{max} and q_i^{last} denote the length of the largest and the last non-preemptive region of τ_i, respectively.

Note that, under non-preemptive scheduling, tasks may share mutually exclusive resources without introducing additional blocking, as long as critical sections are entirely included inside the non-preemptive chunks.

III. Motivational Example

In order to show the benefit of the limited preemption to save energy, let us consider an example with two speeds $s_{min} = 0.5$ and $s_{max} = 1$ and two tasks, τ_1 and τ_2, with the following parameters: $C_1 = 30$, $T_1 = D_1 = 80$, $C_2 = 25$ and $T_2 = D_2 = 200$ (computation times are referred to $s_{nom} = s_{max}$). Tasks are scheduled using Deadline Monotonic and, for the sake of simplicity, preemption costs are considered negligible. The utilization factor at s_{max} is 0.5 and the task set results feasible with fully-preemptive, non-preemptive and limited preemptive models. Switching to s_{min}, the computation times become $C_1 = 60$ and $C_2 = 50$ causing a global utilization $U = 1$. Although using both the

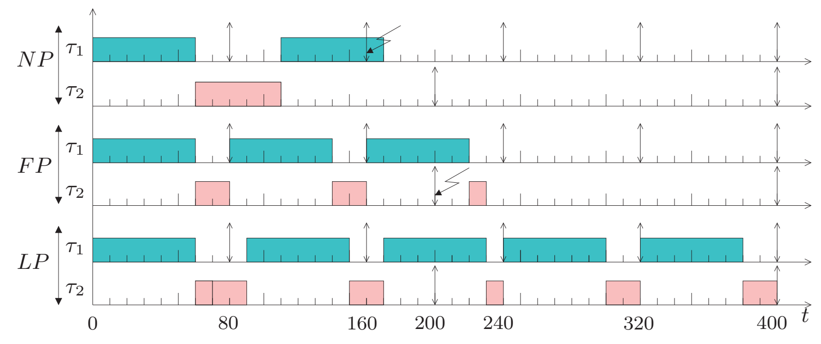


Fig. 1. Schedules at $s = 0.5$ using Non-Preemptive (NP), Fully-Preemptive (FP) and Limited Preemptive (LP) task models.

fully-preemptive and non-preemptive models the task set results unfeasible, a scheduler exploiting the limited preemptive model with deferred preemptions is able to guarantee the task set feasibility (Figure 1). More precisely, with the limited preemptive model, τ_1 consists in a single chuck and τ_2 is split in three chunks of length 10, 20 and 20, respectively.

As a result, using the limited preemption model, the task set can run at s_{min} instead of s_{max} as required by both fully-preemptive and non-preemptive models.

IV. Open Questions

Given the effectiveness of the limited preemption approach to increase schedulability and reduce the processor speed (with respect to fully preemptive algorithms), the open problem to be investigated is then to find good scheduling strategies that leverage limited preemption models to further reduce energy consumption while guaranteeing real-time constraints.

Three possible areas of investigations have been identified, which reflect the main research approaches adopted for energy saving. They are described below.

1) A first step is to develop an algorithm to efficiently compute the slowest processor speed that guarantees real-time constraints. A promising possibility is to extend the algorithm proposed by Bertogna et al. [5] to return not only the set of preemption points, but also the optimal speed.
2) From a DPM point of view, a periodic server could be developed for collecting the idle times together and switch the system into a low-power state during its execution. Since several parameters are involved, the configuration of the server is not trivial. Note that a server running at the highest priority would lead to long continuous intervals spent in a low-power state, but it would increase the interference in lower priority tasks (thus reducing their blocking tolerance). On the other hand, a server running at the lowest priority, would fragment the idle intervals into several slices, preventing an efficient use of DPM techniques.
3) A combination between DPM and DVS techniques could be investigated to find a trade off between the two approaches described above.

References

[1] E. Bini, G. Buttazzo, and G. Lipari, "Speed modulation in energy-aware real-time systems," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, ser. ECRTS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 3–10. [Online]. Available: http://dx.doi.org/10.1109/ECRTS.2005.29

[2] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Adaptive dynamic power management for hard real-time systems," in *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, ser. RTSS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 23–32. [Online]. Available: http://dx.doi.org/10.1109/RTSS.2009.25

[3] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in *Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems*, ser. ECRTS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 92–101. [Online]. Available: http://dx.doi.org/10.1109/ECRTS.2011.17

[4] A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar, "Rate-harmonized scheduling and its applicability to energy management," *IEEE Trans. Industrial Informatics*, vol. 6, no. 3, pp. 265–275, 2010.

[5] M. Bertogna, G. C. Buttazzo, and G. Yao, "Improving feasibility of fixed priority tasks using non-preemptive regions," in *RTSS*, 2011, pp. 251–260.

[6] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited," *Real-Time Systems, Euromicro Conference on*, vol. 0, pp. 269–279, 2007.

[7] C. Maxim, L. Cucu-Grosjean, and O. Zendra, "Towards reducing preemptions to save energy," in *the 5th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2011)*, Nantes, France, Sep. 2011. [Online]. Available: http://hal.inria.fr/hal-00646997

[8] S. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of the 11th Real-Time Systems Symposium (RTSS'90)*, Orlando, Florida, 1990, pp. 182–190.

Heterogeneous multiprocessor compositional real-time scheduling

João Pedro Craveiro and José Rufino
Universidade de Lisboa, Faculdade de Ciências, LaSIGE
Lisbon, Portugal
jcraveiro@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

I. Motivation

Hierarchical scheduling frameworks have recently received a lot of attention within the real-time scheduling community in particular and the research community in general. They have been proposed as a means to cope with different problems, such as the coexistance of hard real-time, soft real-time and best effort workloads in multimedia applications [1], or creating fault containment domains in safety-critical aerospace systems [2]. While traditional approaches focused mainly on two-level hierarchies, more advanced applications created the need for an analysis which would support, in a seamless way, an arbitrary number of levels.

Compositionality is the property of a complex system which can be analysed by recursively analysing its components and the way they are composed. In this sense, a component comprises a workload, a scheduler, and a resource supply. Compositional analysis in hierarchical scheduling frameworks comprises three main points [3].

1) (Local) schedulability analysis: Analysing the schedulability of a component's workload upon its scheduler and resource supply.

2) Component abstraction: Providing an abstract representation for the component's resource demand (hiding the workload's characteristics) as a single real-time requirement identical to a task — dubbed an *interface*. This allows applying classic task schedulers and schedulability analysis to components.

3) Interface composition: Transforming a set of interfaces abstracting the real-time requirements of individual components into an interface abstracting the global requirements of all those components. This is a key point to support hierarchies with arbitrary depths.

II. State of the art

A. Compositional analysis

For the uniprocessor case, most work on compositional analysis revolves around Mok et al.'s bounded-delay resource model [4] and Shin and Lee's periodic resource model [5]. Multiprocessor approaches extending these results include the multiprocessor periodic resource (MPR) model proposed by Shin et al. [6], Bini et al.'s multi supply function (MSF) [7], Bini et al.'s parallel supply function (PSF) [8], and Lipari and Bini's bounded-delay multipartition (BDM) [9]. None of these works explicitly deals with heterogeneous multiprocessors.

We will now focus on the MPR model, since it covers the three main axes of compositional analysis. In [6], a component $\mathcal{C}$ comprises a workload $\tau = \{\tau_i\}_{i=1}^{n}$ of n constrained-deadline sporadic tasks $\tau_i = (C_i, T_i, D_i), C_i \leq D_i \leq T_i, \forall \tau_i \in \tau$, scheduled under global EDF (GEDF) on a cluster of m' identical processors. The proposed MPR model $\Gamma = (\Pi, \Theta, m')$ specifies the provision of Θ units of resource over every period of length Π with concurrency at most m'.

1) Schedulability test: Shin et al. [6] provide a sufficient local schedulability based on a supply bound function for the MPR. The supply bound function $\mathrm{sbf}_\Gamma(t)$ equates to the minimum amount of resource that the MPR $\Gamma = (\Pi, \Theta, m')$ provides over *any* interval with length t.

2) Component abstraction: Based on the presented schedulability test, the authors provide a pseudo-polynomial algorithm to compute the MPR for a component $\mathcal{C}$. For the computation to become tractable, the supply bound function sbf_Γ is replaced by a linear function $\mathrm{lsbf}_\Gamma(t)$ which lower-bounds sbf_Γ.

3) Interface composition: With the MPR, interface composition derives from the transformation of each interface into a periodic task set (since there are no known scheduling algorithms for MPR interfaces). Thus, first, Shin et al. [6] transform each MPR interface $\Gamma = (\Pi, \Theta^*, m^*)$ into a set of m^* periodic tasks $\tau_\Gamma = \{\tau_i = (T_i, C_i, D_i)\}_{i=1}^{m^*}$. These tasks (thus, the components/clusters) can then be scheduled using known algorithms. If the algorithm used is GEDF, the union of the task sets resulting from the transformation of each MPR interface can, in turn, be itself be abstracted with an MPR interface.

B. Global EDF on uniform heterogeneous multiprocessors

Funk et al. [10], Baruah and Goossens [11], and Baruah [12] provide sufficient tests for schedulability of periodic and sporadic tasksets over uniform heterogeneous multiprocessors using the global EDF algorithm with unrestricted migration. Without loss of generality, we will henceforth refer to global EDF with unrestricted migration simply as "global EDF".

III. Open problem

The open problem we here discuss is that of extending virtual cluster-based scheduling to clusters comprising uniform heterogeneous processors, towards compositional hierarchical scheduling frameworks upon heterogeneous multiprocessor platforms. To the best of our knowledge, there is no literature describing compositional hierarchical scheduling frameworks on heterogeneous multiprocessors.

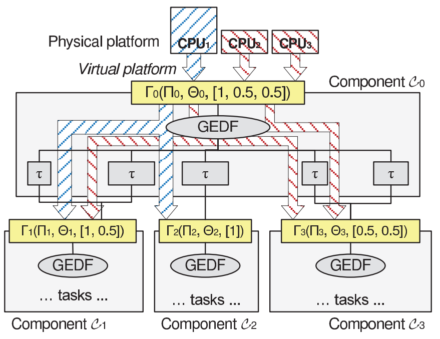


Fig. 1: Heterogeneous compositional framework with the HMPR model

IV. Preliminary intuitions

To solve the described problem, we propose the heterogeneous multiprocessor periodic resource model. A *heterogeneous multiprocessor periodic resource* (HMPR) model $\widetilde{\Gamma} = (\Pi, \Theta, \pi)$ specifies the provision of Θ units of resource over every period of length Π over a virtual cluster $\pi = \{s_i''\}_{i=1}^{m''}$ comprising m'' heterogeneous processors. Processors are represented as normalized relative speeds, such that $1.0 \geq s_i \geq s_{i+1} > 0.0, \forall i < m''$. For the purpose of establishing connections with the work of Shin and Lee [6], let us note that an MPR $\Gamma = (\Pi, \Theta, m')$ translates to an HMPR $\widetilde{\Gamma} = (\Pi, \Theta, \{s_i' = 1.0\}_{i=1}^{m'})$. Other than this, our system model is the same, as described in Section II-A.

Figure 1 provides a graphical representation of the kind of hierarchy the problem and its solution shall be laid upon. A root component, $\mathcal{C}_0$, receives a virtual resource provision directly from the physical platform, whereas the remaining components receive their virtual resource provision from $\mathcal{C}_0$.

Our initial intuition was that, if we consider only GEDF scheduling, then the results of [6] up to (and partially including) component abstraction are applicable. We were meanwhile able to formalize and prove this intuition as follows.

Lemma 1. *Let $\Gamma = (\Pi, \Theta, m')$ be the MPR interface abstracting a component $\mathcal{C}$ comprising a task set τ scheduled under global EDF on a virtual cluster comprising m' identical processors. If τ is schedulable using Γ, then τ is schedulable using any HMPR interface $\widetilde{\Gamma} = (\Pi, \Theta, \{s_i''\}_{i=1}^{m''})$, such that $\sum_{i=1}^{m''} s_i'' \geq m'$.*

Proof sketch: The only difference between the considered MPR and HMPR is the virtual cluster (or platform) upon which tasks are scheduled. Let $\pi' = \{s_i' = 1.0\}_{i=1}^{m'}$ represent the MPR's platform (upon which we know τ is GEDF-schedulable) and $\pi'' = \{s_i''\}_{i=1}^{m''}$ the HMPR's platform. Since these platforms fulfil the conditions of Lemma 1 of [10], and GEDF is a work-conserving algorithm, τ is GEDF-schedulable on π''. ■

V. Open questions

Progressing from the result in Lemma 1, further questions are open for this problem.

Component abstraction: How do we select π'' over the various alternatives which fulfil Lemma 1? Can we tighten the analysis in [6] by deriving an HMPR-specific supply bound function?

Interface composition: How do we transform an HMPR interface $\widetilde{\Gamma} = (\Pi, \Theta, \{s_i''\}_{i=1}^{m''})$ into m'' periodic tasks to be scheduled under GEDF? Since this transformation must take into account the different relative speeds of the processors in the virtual cluster, how do we guarantee that each of these tasks will, in the end, be scheduled upon the right processor in the physical heterogeneous platform? How do we compose HMPRs?

Adopted abstraction: Could other interfaces (MSF [7], PSF [8], BDM [9]) bring more advantage in being employed to support heterogeneous multiprocessor platforms? Could the HMPR be based on a simpler representation of the platform (e.g., only total capacity and λ parameter [12] instead of individual processor speeds), in favour of enhanced composability?

VI. Summary

We have described the problem of compositional hierarchical scheduling frameworks upon heterogeneous multiprocessor platforms. For this, we suggest extending virtual cluster-based scheduling to clusters comprising heterogeneous processors, and we introuce the hierarchical multiprocessor periodic resource (HMPR) model. Our starting point is that some results on component abstraction obtained for the multiprocessor periodic resource (MPR) model by Shin et al. [6] apply with due adaptation, with new results being needed to complete and tighten the component abstraction, solve the interface composition problem, and answer other open questions.

Acknowledgments: This work was partially supported by FCT, through the Multiannual and CMU|Portugal programs, and the Individual Doctoral Grant SFRH/BD/60193/2009.

The authors would also like to thank Insik Shin for being available to answer our doubts about specific aspects of MPR interface composition in [6].

References

[1] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *RTSS '98*, Madrid, Spain, Dec. 1998.
[2] J. Rufino, J. Craveiro, and P. Verissimo, "Architecting robustness and timeliness in a new generation of aerospace systems," in *Architecting Dependable Systems VII*, ser. LNCS, A. Casimiro, R. de Lemos, and C. Gacek, Eds. Springer, 2010, vol. 6420, pp. 146–170.
[3] I. Shin and I. Lee, "Compositional real-time schedulability analysis," in *Handbook of Real-Time and Embedded Systems*, ser. Computer and Information Science series, I. Lee, J. Y.-T. Leung, and S. H. Son, Eds. Chapman & Hall / CRC, 2007.
[4] A. K. Mok, X. A. Feng, and D. Chen, "Resource partition for real-time systems," in *RTAS '01*, Taipei, Taiwan, May/Jun. 2001.
[5] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS '03*, Cancun, Mexico, Dec. 2003.
[6] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *ECRTS '08*, Prague, Czech Republic, Jul. 2008.
[7] E. Bini, G. Buttazzo, and M. Bertogna, "The multi supply function abstraction for multiprocessors," in *RTCSA '09*, Beijing, China, Aug. 2009.
[8] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in *RTSS '09*, Washington, D.C., Dec. 2009.
[9] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation," in *RTSS '10*, Nov./Dec. 2010.
[10] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," in *RTSS '01*, London, UK, Dec. 2001.
[11] S. Baruah and J. Goossens, "The EDF scheduling of sporadic task systems on uniform multiprocessors," in *RTSS '08*, Barcelona, Spain, Nov./Dec. 2008.
[12] S. Baruah, "An improved global EDF schedulability test for uniform multiprocessors," in *RTAS '10*, Stockholm, Sweden, Apr. 2010.

NOTES