On the Homogeneous Multiprocessor Virtual Machine Partitioning Problem

Stefan Groesbrink

On the Homogeneous Multiprocessor Virtual Machine Partitioning Problem

Stefan Groesbrink

Design of Distributed Embedded Systems, Heinz Nixdorf Institute
University of Paderborn
Fuerstenallee 11, 33102 Paderborn, Germany
s.groesbrink@upb.de, http://www.hni.uni-paderborn.de/

Abstract. This work addresses the partitioning of virtual machines with real-time requirements onto a multi-core platform. The partitioning is usually done manually through interactions between subsystem vendors and system designers. Such a proceeding is expensive, does not guarantee to find the best solution, and does not scale with regard to the upcoming higher complexity in terms of an increasing number of both virtual machines and processor cores. The partitioning problem is defined in a formal manner by the abstraction of computation time demand of virtual machines and computation time supply of a shared processor. The application of a branch-and-bound partitioning algorithm is proposed. Combined with a generation of a feasible schedule for the virtual machines mapped to a processor, it is guaranteed that the demand of a virtual machine is satisfied, even if independently developed virtual machines share a processor. The partitioning algorithm offers two optimization goals, required number of processors and the introduced optimization metric criticality distribution, a first step towards a partitioning that considers multiple criticality levels. The different outcomes of the two approaches are illustrated exemplarily.

1 Introduction

This work targets the hypervisor-based integration of multiple systems of mixed criticality levels on a multicore platform. System virtualization refers to the division of the resources of a computer system into multiple execution environments in order to share the hardware among multiple operating system instances. Each guest runs within a virtual machine—an isolated duplicate of the real machine. System virtualization is a promising software architecture to meet many of the requirements of complex embedded systems and cyber-physical systems, due to its capabilities such as resource partitioning, consolidation with maintained isolation, transparent use of multiple processor system-on-chips, and cross-platform portability.

The rise of multi-core platforms increases the interest in virtualization, since virtualization's architectural abstraction eases the migration to multi-core platforms [11]. The replacement of multiple hardware units by a single multi-core

system has the potential to reduce size, weight, and power. The coexistence of mixed criticality levels has been identified as one of the core foundational concepts for cyber-physical systems [3]. System virtualization implies it in many cases, since the applicability of virtualization is limited significantly if the integration of systems of different criticality level is not allowed.

Contribution This work addresses the partitioning of virtual machines with real-time requirements onto a multi-core platform. We define this design problem as the *homogeneous multiprocessor virtual machine partitioning problem* in a formal manner, specifying the computation time demand of virtual machines and the computation time supply of a shared processor. A mapping of a given set of virtual machines among a minimum number of required processors is achieved by a branch-and-bound algorithm, such that the capacity of any individual processor is not exceeded. This automated solution provides analytical correctness guarantees, which can be used in system certification. An introduced optimization metric is a first step towards a partitioning that considers multiple virtual machine criticality levels appropriately.

2 System Model

2.1 Task Model and Virtual Machine Model

According to the *periodic task model*, each periodic task τ_i is defined as a sequence of jobs and characterized by a period T_i, denoting the time interval between the activation times of consecutive jobs [16]. The worst-case execution time (WCET) C_i of a task represents an upper bound on the amount of time required to execute the task. The utilization $U(\tau_i)$ is defined as the ratio of WCET and period: $U(\tau_i) = C_i/T_i$. A criticality level χ is assigned to each task [24]. Only two criticality levels are assumed in this work, HI and LO.

A virtual machine V_k is modeled as a set of tasks and a scheduling algorithm A, which is applied by the guest operating system. A criticality level χ is assigned to each virtual machine. If a virtual machine's task set is characterized by multiple criticality levels, the highest criticality level determines the criticality of the virtual machine.

2.2 Multi-core and Virtual Processor

Target platform are homogeneous multi-core systems, consisting of m identical cores of equal computing power. This implies that each task has the same execution speed and utilization on each processor core. Assumed is in addition a shared memory architecture with a uniform memory access.

A virtual processor is a representation of the physical processor to the virtual machines. A dedicated virtual processor P_k^{virt} is created for each virtual machine V_k. It is in general slower than the physical processor core to allow a mapping of multiple virtual processors onto a single physical processor core. A virtual

processor is modeled as a processor capacity reserve [18], a function $\Pi(t)\colon \mathbb{N} \mapsto \{0,1\}$ defined as follows:

$$\Pi(t) = \begin{cases} 0, & \text{resource not allocated} \\ 1, & \text{resource allocated} \end{cases} \tag{1}$$

The computation capacity of a physical processor core is partitioned into a set of reservations. Each reservation is characterized by a tuple (Q_k, Υ_k): in every period Υ_k, the reservation provides Q_k units of computation time. $\alpha_k = Q_k/\Upsilon_k$ denotes the bandwidth of the virtual processor.

The computational service provided by a virtual processor P_k^{virt} can be analyzed with its supply function $Z_k(t)$, as introduced by Mok et al. [19]. $Z_k(t)$ returns the minimum amount of computation time (worst-case) provided by the virtual processor in an arbitrary time interval of length $t \geq 0$:

$$Z_k(t) = \min_{t_0 \geq 0} \int_{t_0}^{t_0+t} \Pi(x)dx \ . \tag{2}$$

2.3 Notation

The symbols in this paper are therefore defined as follows:

1. $\tau_i = (C_i, T_i)$: task i with WCET C_i, period T_i, utilization $U(\tau_i) = C_i/T_i$
2. $P = \{P_1, P_2, ..., P_m\}$: set of processors ($m \geq 2$)
3. $V = \{V_1, V_2, ..., V_n\}$: set of virtual machines ($n \geq 2$)
4. $\tau_i \in V_k$: task τ_i is executed in V_k
5. $U(V_k) = \sum_{\tau_i \in V_k} U(\tau_i)$: utilization of V_k
6. $\chi(V_k) \in \{LO, HI\}$: criticality level of V_k
7. $P^{virt} = \{P_1^{virt}, P_2^{virt}, ..., P_n^{virt}\}$: virtual processors, V_k is mapped to P_k^{virt}
8. (Q_k, Υ_k) : resource reservation with bandwidth $\alpha_k = Q_k/\Upsilon_k$
9. $Z_k(t)$: minimum amount of computation time provided by P_k^{virt}
10. $\Gamma(P_i)$: subset of virtual processors allocated to P_i

All parameters of the system—number of processors and computing capacity, number of virtual machines and parameters of all virtual machines, number of tasks and parameters of all tasks—are a priori known.

3 The Homogeneous Multiprocessor Virtual Machine Partitioning Problem

The scheduling problem for system virtualization on multi-core platforms consists of two sub-problems:

(i) partitioning: mapping of the virtual machines to processor cores
(ii) uniprocessor hierarchical scheduling on each processor core

Sub-problem (ii) is well-understood and many solutions are available, e.g. [15]. This work focuses on sub-problem (i) and refer to it as the homogeneous multiprocessor virtual machine partitioning problem. More precisely, the virtual processors P^{virt} executing the virtual machines V have to be mapped to the physical processors P:

$$V \overset{f_1}{\mapsto} P^{virt} \overset{f_2}{\mapsto} P \ . \tag{3}$$

f_1 is a bijective function: each virtual machine V_k is mapped to a dedicated virtual processor P_k^{virt}. f_2 maps 0 to $n = |P^{virt}|$ virtual processors to each element of P. A solution to the problem is a partition Γ, defined as:

$$\Gamma = (\Gamma(P_1), \Gamma(P_2), ..., \Gamma(P_m)) \tag{4}$$

Such a mapping of virtual machines (equivalent to virtual processors) to physical processors is correct, if and only if the computation capacity requirements of all virtual processors are met; and by consequence the schedulability of the associated virtual machines is guaranteed.

The partitioning problem is equivalent to a bin-packing problem, as for example Baruah [2] has shown for the task partitioning problem by transformation from 3-Partition. The virtual machines are the objects to pack with size determined by their utilization factors. The bins are processors with a computation capacity value that is dependent on the applied virtual machine scheduler of this processor. The bin-packing problem is known to be intractable (NP-hard in the strong sense) [10] and the research focused on approximation algorithms [6].

4 Scheduling Scheme

It is an important observation that the hypervisor-based integration of independently developed and validated systems implies partitioned scheduling. As a coarse-grained approach, it consolidates entire software stacks including an operating system, resulting in scheduling decisions on two levels (hierarchical scheduling). The hypervisor schedules the virtual machines and the hosted guest operating systems schedule their tasks according to their own local scheduling policies. This is irreconcilable with a scheduling based on a global ready queue.

Virtual Machine Scheduling In the context of this work, n virtual machines are statically assigned to $m < n$ processors. Although a dynamic mapping is conceptually and technically possible, a static solution eases certification significantly, due to the lower run-time complexity, the higher predictability, and the wider experience of system designer and certification authority with uniprocessor scheduling. Run-time scheduling can be performed efficiently in such systems and the overhead of a complex virtual machine scheduler is avoided.

For each processor, the virtual machine scheduling is implemented based on fixed time slices. Execution time windows within a repetitive major cycle are assigned to the virtual machines based on the required utilization and the maximum blackout time. As a formal model, the *Single Time Slot Periodic Partitions*

model by Mok et al. [20] is applied. A resource partition is defined as N disjunct time intervals $\{(S_1, E_1), ..., (S_N, E_N)\}$ and a partition period $P_{partition}$, so that a virtual machine V_i is executed during intervals $(S_i + j \cdot P_{partition}, E_i + j \cdot P_{partition})$ with $j \geq 0$. Kerstan et al. [13] presented an approach to calculate such time intervals for virtual machines scheduled by either earliest deadline first (EDF) or rate-monotonic (RM), with $S_0 = 0$ and $S_i = E_{i-1}$:

$$E_i = \begin{cases} S_i + U(V_i) \cdot P_{partition} & \text{in case of EDF} \\ S_i + \frac{1}{U_{lub}^{RM}(V_i)} U(V_i) \cdot P_{partition} & \text{in case of RM} \end{cases}, \text{ with} \tag{5}$$

$$U_{lub}^{RM}(V_i) = n_{tasks} \cdot (2^{\frac{1}{n_{tasks}}} - 1) \tag{6}$$

In case of RM, a scaling relative to the least upper bound U_{lub}^{RM} is required. If the partition period is chosen as $P_{partition} = \gcd(\{T_k | \tau_k \in \bigcup_{i=1}^{n} V_i\})$, no deadline will be missed [13].

The virtual machine schedule is computed offline and stored in a dispatching table, similar to the cyclic executive scheduling approach [1]. The size of this table is bounded, since the schedule repeats itself after $P_{partition}$. Such a highly predictable and at design time analyzable scheduling scheme is the de facto standard for scheduling high-criticality workloads [21].

In the terms of the resource reservation model of the virtual processor, the bandwidth α_k of the virtual processor P_k^{virt} that executes virtual machine V_k is equal to $(E_k - S_k)/P_{partition}$, with $\Upsilon_k = P_{partition}$ and $Q_k = \alpha \cdot \Upsilon$. Note that this abstraction of the computation time demand of a virtual machine to a recurring time slot that is serviced by a virtual processor (Q_k, Υ_K) allows to regard the virtual machine as a periodic task and transforms the virtual machine partitioning problem to the task partitioning problem.

Task Scheduling Any scheduling algorithm can be applied as task scheduler, as long as it allows to abstract the computation time requirements of the task set in terms of a demand-bound function $dbf(V_i, t)$, which bounds the computation time demand that the virtual machine could request to meet the timing requirements of its tasks within a specific time interval of length t [23]. As a task set cannot possibly be schedulable according to any algorithm if the total execution that is released in an interval and must also complete in that interval exceeds the available processing capacity, the processor load provides a simple necessary condition for taskset feasibility:
A virtual machine V_k, applying A as local scheduler and executed by a virtual processor P_k^{virt} characterized by the supply function Z_k, is schedulable if and only if $\forall\ t > 0 : dbf_A(V_k, t) \leq Z_k(t)$ (compare [23]).

5 Partitioning Algorithm

Common task set partitioning schemes apply Bin-Packing Heuristics or Integer-Linear-Programming (ILP) approaches in order to provide an efficient algorithm

[5][7]. In the context of this work, however, the number of virtual machines is comparatively small and the partitioning algorithm is to be run offline and does not have to be executed on the embedded processor. Therefore, the algorithm performs a systematic enumeration of all candidate solutions following the branch-and-bound paradigm [14]. The depth of the search tree is equal to the number of virtual machines n.

Two optimization goals are considered, according to which candidates are compared. Minimizing the number of processors is the basic optimization goal. In addition, the goal can be set to maximize the *CriticalityDistribution*, a metric defined as follows:

Definition. The *CriticalityDistribution* Z denotes for a partitioning Γ the distribution of the $n_{crit} \leq n$ HI-critical virtual machines among the m processors:

$$Z(\Gamma) = \frac{\sum_{i=1}^{m} \zeta(P_i)}{n_{crit}} \quad \text{, with} \tag{7}$$

$$\zeta(P_i) = \begin{cases} 1, & \text{if } \exists P_j^{virt} \in \Gamma(P_i) \colon \chi(V_j) = HI, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

For example, assumed that $n_{crit} = 4$ and $m = 4$, Z equals 1 if there is at least one HI-critical virtual machine mapped to all processors; and Z equals 0.75 if one processor does not host a HI-critical virtual machine. This results, if the maximum number of processors is not limited, to a mapping of each cirtical virtual machine to a dedicated processor, which is potentially shared with LO-critical virtual machines, but not with other HI-critical virtual machines.

The motivation of the optimization goal criticality distribution is the *Criticality Inversion Problem*, defined by de Niz et al. [8]. Transferred to virtual machine scheduling, criticality inversion occurs if a HI-critical virtual machine overruns its execution time budget and is stopped to allow a LO-critical virtual machine to run, resulting in a deadline miss for a task of the HI-critical virtual machine. By definition of criticality, it is more appropriate to continue the execution of the HI-critical virtual machine, which can be done for highly utilized processors by stealing execution time from the budget of LO-critical virtual machines. It is in general easier to avoid criticality inversion, if virtual machines of differing criticality share a processor. If the number of virtual machines does not exceed the number of physical processors, all critical virtual machine are mapped to different physical processors. The partitioning algorithm either minimizes the number of processors or maximizes the criticality distribution, while minimizing the number of processors among partitions of same criticality distribution.

Before generating the search tree, the set of virtual machines V is sorted according to decreasing utilization. This is motivated by a pruning condition: if at some node, the bandwidth assigned to a processor is greater than 1, the computational capacity of the processor is overrun and the whole subtree can be pruned. Such a subtree pruning tends to occur earlier, if the virtual machines are ordered according to decreasing utilization.

We introduce that a virtual machine is termed to be *heavy*, if certification requires that this virtual machine is exclusively mapped to a dedicated processor

or if other virtual machines can only be scheduled in background, i.e. the heavy virtual machine is executed immediately whenever it is has a computation demand. By consequence, a heavy virtual machine cannot be mapped to the same processor as other HI-critical virtual machines.

6 Example

The different outcome dependent on the optimization goal of the algorithm is illustrated with the examplary virtual machine set of Table 1. EDF is assumed for all virtual machines, so that a scaling is not required and $\alpha_k = U(V_k)$.

Table 1. Example: Set of Virtual Machines

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
χ	LO	HI	LO	LO	HI	HI	HI	LO	LO	HI
U	0.6	0.5	0.5	0.3	0.25	0.2	0.2	0.2	0.2	0.15

Figure 1 depicts the virtual machine to processor mapping for three different goals, with a red virtual machine identifier denoting a HI-critical virtual machine. Subfigure (a) depicts the outcome for the optimization of the number of processors. The virtual machine set is not schedulable on less than four processors. The average utilization per processor is 0.775 and the criticality distribution Z is $3/5 = 0.6$. Subfigure (b) depicts the outcome for the optimization of the criticality distribution, however with a maximum number of $m_{max} = 4$ processors allowed. The allocation is therefore still characterized by the minimum number of processors. The criticality distribution Z improves to $4/5 = 0.8$. From a criticality point of view, this mapping is more suitable, since the options to avoid criticality inversion on processor P_3 are very limited in the first solution. Subfigure (c) depicts an unrestricted optimization of the criticality distribution, resulting in an additional processor. The optimal criticality distribution $Z = 1$ is achieved, however at the cost of exceeding the minimum number of processors, which leads to a decrease of the average utilization per processor to 0.62. The last mapping is the correct choice, if the five HI-critical virtual machines are heavy.

7 Related Work

The related problem of partitioning a periodic task set upon homogeneous multiprocessor platforms has been extensively studied, both theoretically and empirically [5][7]. Lopez et al. observed that ordering tasks according to decreasing utilization prior to the partitioning proves helpful [17], a technique applied in this work as well. Buttazzo et al. proposed a branch-and-bound algorithm for

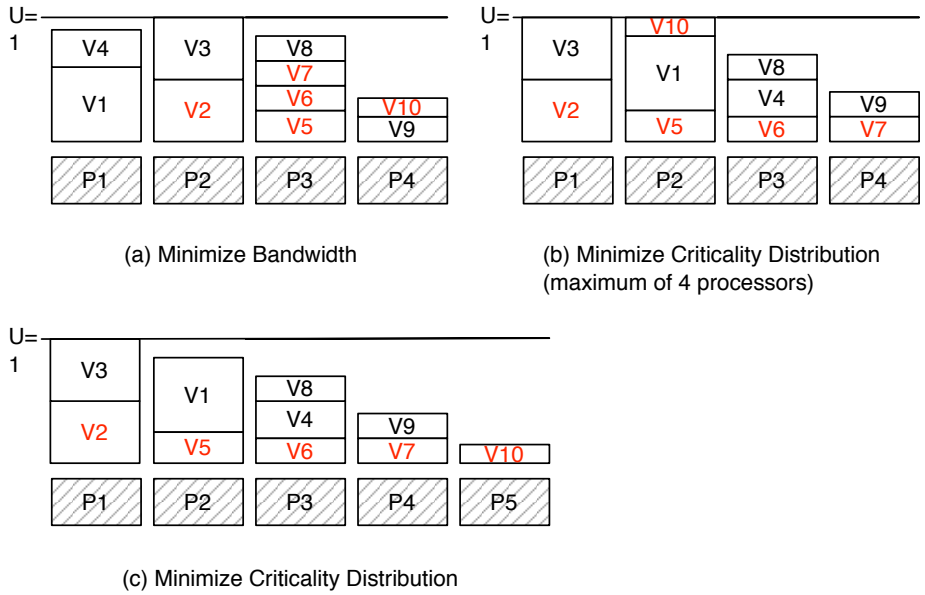

Fig. 1. Mappings for different Optimization Goals

partitioning a task set with precedence constraints, in order to minimize the required overall computational bandwidth [4]. Peng and Shin presented a branch-and-bound algorithm in order to partition a set of communicating tasks in a distributed system [22].

Kelly at al. proposed bin-packing algorithms for the partitioning of mixed-criticality real-time task sets [12]. Using a common mixed-criticality task model (characterized by an assignment of multiple WCET values, one per each criticality level in the system), they experimentally compare different kinds of task ordering according to utilization and criticality and observed that the latter solutions results in a higher percentage of finding a feasible schedule for a randomly generated task set.

Shin and Lee introduced a formal description of the component abstraction problem (abstract the real-time requirements of a component) and the component composition model (compose independently analyzed locally scheduled components into a global system)[23]. Easwaran et al. introduced compositional analysis techniques for automated scheduling of partitions and processes in the specific context of the ARINC-653 standard for distributed avionics systems [9], however, did not tackle the mapping of partions to processors. As required by the ARINC specification and as done in this work, a static partition schedule is generated at design time. Both the partitions and the tasks within the partitions are scheduled by a deadline-monotonic scheduler.

8 Conclusion and Future Work

This work defined the partitioning problem of mapping virtual machines with real-time constraints to a homogeneous multiprocessor architecture in a formal manner. This is the prerequisite for an algorithmic solution. Formal models were adapted to abstract and specify the computation time demand of a virtual machine and the computation time supply of a shared processor, in order to analytically evaluate whether it is guaranteed that the demand of a virtual machine is satisfied. The application of a branch-and-bound algorithm is proposed with two optimization metrics. A brief introduction on how to generate a feasible virtual machine schedule after the partitioning was given. A highly predictable and at design time analyzable scheduling scheme based on fixed time slices was chosen as this is the de facto standard for scheduling high-criticality systems.

Partitioning and schedule generation together guarantee that all virtual machines obtain a sufficient amount of computation capacity and obtain it in time, so that the hosted guest systems never miss a deadline. This automated solution provides analytical correctness guarantees, which can help with system certification. In contrast to a manual partitioning, it guarantees to find the optimal solution and scales well with regard to an increasing number of both virtual machines and processor cores. The optimization metric criticality distribution is a first step towards a partitioning that considers multiple criticality levels appropriately. The different outcomes of the two approaches were illustrated exemplarily.

The presented algorithm serves as a groundwork for a research of the partitioning problem. In particularly, we are going to include the overhead of virtual machine context switching, since it is for most real implementations extensive enough to not be neglected. The partitioning directly influences the virtual machine scheduling, which in turn heavily influences the number of virtual machine context switches. In addition, communication between virtual machines should be included, since the communication latency depends on the fact whether two virtual machines share a core or not. A further interesting question is whether a more detailled analysis of the timing characteristics of the virtual machines, in order to map guests with similar characteristics to the same processor, leads to better results.

Acknowledgments. This work was funded within the project ARAMiS by the German Federal Ministry for Education and Research with the funding IDs 01IS11035. The responsibility for the content remains with the authors.

References

1. Baker, T., Shaw, A.: The cyclic executive model and ada. Real-Time Systems (1989)
2. Baruah, S.: Task partitioning upon heterogeneous multiprocessor platforms. In: Real-Time and Embedded Technology and Applications Symposium (2004)

3. Baruah, S., Li, H., Stougie, L.: Towards the design of certifiable mixed-criticality systems. In: Real-Time and Embedded Technology and Applications Symposium (2010)
4. Buttazzo, G., Bini, E., Wu, Y.: Partitioning real-time applications over multi-core reservations. In: IEEE Transactions on Industrial Informatics. vol. 7, pp. 302–315 (2011)
5. Carpenter, J. et al.: A categorization of real-time multiprocessor scheduling problems and algorithms. In: Handbook on Scheduling Algorithms, Methods, and Models (2004)
6. Coffman, E., Garey, M., Johnson, D.: Approximation algorithms for bin packing: a survey. In: Approximation algorithms for NP-hard problems. pp. 46—93 (1996)
7. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. In: ACM Computing Surveys (2010)
8. de Niz, D. et al.: On the scheduling of mixed-criticality real-time task sets. In: Real-Time Systems Symposium (2009)
9. Easwaran, A. et al.: A compositional scheduling framework for digital avionics systems. In: Real-Time Computing Systems and Applications (2009)
10. Garey, M., Johnson, D.: Computers and Intractability. W.H. Freman, New York (1979)
11. Intel Corporation (White paper): Applying multi-core and virtualization to industrial and safety-related applications. http://download.intel.com/platforms/applied/indpc/321410.pdf (2009)
12. Kelly, O., Aydin, H., Zhao, B.: On partitioned scheduling of fixed-priority mixed-criticality task sets. In: IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (2011)
13. Kerstan, T., Baldin, D., Groesbrink, S.: Full virtualization of real-time systems by temporal partitioning. In: Workshop on Operating Systems Platforms for Embedded Real-Time Applications (2010)
14. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica 28(3), 497—520 (1960)
15. Lipari, G., Bini, E.: Resource partitioning among real-time applications. In: Euromicro Conference on Real-Time Systems (2003)
16. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. In: Journal of the ACM (1973)
17. Lopez, J., Garcia, M., Diaz, J., Garcia, D.: Utilization bounds for multiprocessor rate-monotonic systems. Real-Time Systems (2003)
18. Mercer, C. et al.: Processor capacity reserves: Operating system support for multimedia applications. In: Multimedia Computing and Systems (1994)
19. Mok, A., Feng, A.: Real-time virtual resource: A timely abstraction for embedded systems. In: Lecture Notes in Computer Science. vol. 2491, pp. 182–196 (2002)
20. Mok, A., Feng, X., Chen, D.: Resource partition for real-time systems. In: Real-Time Technology and Applications Symposium (2001)
21. Mollison, M. et al.: Mixed-criticality real-time scheduling for multicore systems. In: International Conference on Computer and Information Technology (2010)
22. Peng, D., Shin, K.: Assignment and scheduling communicating periodic tasks in distributed real-time systems. IEEE Transactions on Software Engineering (1997)
23. Shin, I., Lee, I.: Compositional real-time scheduling framework with periodic model. In: ACM Transactions on Embedded Computing Systems. vol. 7 (2008)
24. Vestal, S.: Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proc. of the Real-Time Systems Symposium (2007)