Continuous Performance Analysis of Fault-Tolerant Virtual Machines *

Boguslaw Jablkowski and Olaf Spinczyk
Technische Universität Dortmund
Embedded System Software Group
{boguslaw.jablkowski,olaf.spinczyk}@tu-dortmund.de

Abstract: Virtual machine technology has been successfully applied for the construction of fault-tolerant computing systems. For example, vmware Fault Tolerance and Xen Remus support transparent failover of VMs running on different physical machines in a local area network. However, high availability alone is in many application domains not sufficient. Especially in the context of Cyber-Physical Systems, which interact with the physical environment, realtime constraints have to be fulfilled in order to avoid damage. Therefore, we are working on the combination of VM-based fault tolerance with a performance analysis technique, namely the modular performance analysis with real-time calculus. Such enhanced system would at any time be aware of its own performance and could use this information for smarter reconfiguration decisions in case of faults. This paper will sketch the underlying model, the envisioned system architecture, and discuss beneficial application scenarios.

1 Introduction

The need of constant optimization and rationalization leads computer systems designers and architectures to continuously search for solutions that not only fulfill the functional requirements of their domain problems, but are also capable of meeting the non-functional expectations. Among them are infrastructural and maintenance cost reduction, dynamic reconfiguration options, fault-tolerance solutions and not uncommonly the need of satisfying real-time constraints. Many of these requirements can already be satisfied through an relatively old technique, known as virtualization.

In recent years the popularity of this approach has noticeably grown. The main reason for this is the rising computational power of modern computers. As a consequence of this fact, the spectrum of applications for virtualizasion-based techniques is constantly expanding to new domains. An example of such a new area of applications, where the above mentioned (non-)functional requirements have to be strictly fulfilled, are monitoring, protection and control systems for prospective electrical transmission systems. In this

*This work was partly supported by the German Research Foundation (DFG) under grant no. SP 968/5-1 and SP968/6-1.

sector, due to liberalization of energy markets and the integration of renewable energy sources, new challenges have emerged. Novel concepts of execution and communication platforms for power system management have to be researched and evaluated in order to meet this challenges.

However, power management systems are not the only sophisticated systems that could profit from virtualization-based solutions. In this paper we introduce an envisioned architecture, which we believe is capable of providing the necessary infrastructure for realizing such demanding and complex systems. The novelty of our approach lies in the combination of virtualization-based fault-tolerance real-time systems with a continuous performance analysis. The latter provides us with information, which in turn can be used for smart reconfiguration decisions in order to sustain critical applications and services in case of faults.

The rest of this paper is structured as follows. Section 2 provides the related work. In section 3, we introduce the modular performance analysis with real-time calculus. Section 4 presents our envisioned system architecture. In section 5, we describe two abstract application scenarios. Section 6 discusses some of the challenges our approach rises. Finally, we conclude our work in section 7.

2 Related Work

To the best of our knowledge, no other approach has combined continuous performance analysis with virtualization in order to allow fault-tolerant real-time systems. Nonetheless, in the past, virtualization has been successfully used for constructing fault-tolerant systems. Several works have been dedicated to this topic.

A pioneer system for hypervisor-based fault-tolerance was proposed by Bressoud and Schneider [BS96]. It introduced high availability through lock-stepped replication of applications. In this approach, the input events from the primary host are being forwarded to the backup host and deterministically replayed. Alternative solutions for high-availability are Remus [CLM+08] and Kemari [TKSS08]. Instead of running two hosts in lock-step mode, Remus allows speculative execution and asynchronous replication. Performance optimization techniques for the checkpoint-recovery mechanisms have been recently discussed in [ZDJ+10]. Another approach for hypervisor-based fault tolerant systems is the RESH architecture [RHKSP06]. It allows to tolerate, on a single host, non-benign random faults and software faults by using redundant execution of a service and N-version programming [AC77]. This approach was extended to multiple hosts in [RK07].

Recently, works have been introduced, which aim at bridging the gap between real-time systems and virtualization. In [LXC+12] the authors propose a framework for realizing

compositional scheduling of real-time systems through virtualization. In [XWLG11], a real-time scheduling hypervisor extension for Xen [BDF+03], called RT-Xen, was presented. RT-Xen provides an open-source platform to evaluate real-time scheduling techniques and offers a suite of fixed-priority servers for empirical study. In [BTS+10] the authors examine real-time virtualization using the L4/Fiasco as hypervisor and L4Linux as guest operating systems.

There are several different approaches to performance analysis. Differences between these techniques include - among others - their evaluation time, modeling scope and accuracy. For the reason of modularity, short evaluation times and tool support, we chose the modular performance analysis with real-time calculus. The following introduction to this method is based upon [WTVL06] and [HHB+12]. An evaluation and comparison of different performance analysis techniques can be found in [Per11].

3 Modular Performance Analysis and Real-Time Calculus

Modular performance analysis (MPA) with real-time calculus (RTC) is a formal technique for modeling and analysis of distributed systems. Modular performance analysis is based on the Y-chart system design scheme [KDVW97] and embeds real-time calculus to compute hard lower and upper bounds to delays and buffer requirements. It is therefore suited for the performance analysis of hard real-time systems. The real-time calculus is a specialization of the network calculus [Cru91, LT01], which is a theoretical method used to analyze computer networks. Both, the network calculus and the real-time calculus, are based on the max-plus algebra [BCOQ92]. The unification of modular performance analysis and real-time calculus has been proposed in [CKT03] and further developed in [Wan06].

MPA-RTC follows a modular and interface-based approach. This allows to independently characterize system components and interconnect them in order to analyze their global interference. MPA-RTC is based on events, resources, workload and component models. The first describes how often system function or jobs are invoked by the environment. The resource model provides information about the computational and communication capacity of the hardware. The workload model specifies the execution or communication demands that functions or jobs impose on a system resource. Finally, the component model defines the processing semantics for a task mapped to a component. The information needed to specify these models have to be determined otherwise. Possible information sources are data sheets, reference manuals and well-established techniques like WCET analysis or simulation based approaches. In the following a short description of the above mentioned models is given.

3.1 Event Model

Events are often used to model the behavior of the system's environment. They are responsible for triggering computational or communication processes in a system. In MPA-RTC, the event stream is described as a tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ of functions, also called *arrival curves*, of which the first describes the upper and the latter the lower bound of events that arrived in any time interval of a specified length. In other words, $\alpha^u(\Delta)$ specifies the maximum and $\alpha^l(\Delta)$ the minimum amount of incoming events within the time interval Δ.

3.2 Resource Model

The capacity of the computation and communication hardware within the system is described analogously to the arrival curve's functions. A tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of functions specifies the upper and the lower bound of available service capacity within the time interval Δ. The functions $\beta^u(\Delta)$ and $\beta^l(\Delta)$ are also called *service curves* and provide an abstract resource model of the system. In MPA-RTC resources can be scheduled with different policies. Among others, supported are (non-)preemptive fixed priority scheduling (FP), time division multiple access (TDMA) and earliest deadline first (EDF).

3.3 Workload Model

In order to compute the remaining resource capacity and the outgoing event stream, the execution demands for the incoming events have to be known. Further, arrival and service curves have to be expressed in the same unit. For this purpose another formalism is being used, the *workload curves*. A tupel $\gamma(e) = [\gamma^u(e), \gamma^l(e)]$ of functions specifies the upper and lower workload bounds for a given number of succeeding events. The upper and lower bounds can be interpreted as the worst-case (WCET) and best-case (BCET) execution time.

3.4 Component Model

In a system, incoming events are being processed by a number of tasks, all of which are mapped to components. In MPA-RTC, processing components are modeled as *greedy processing components* (GPC). Their semantics is as follows: Events that arrive at a GPC are represented as arrival curves and stored in a FIFO input buffer. Next, for every event, a task is being instantiated and processed by the GPC. The processing power is restricted by the available capacity of the resource, which is described by the upper and lower service curves. The output of the GPC is again represented as arrival and service curves, which in turn can be used as input for the next component. Additionally, for every GPC the maximum processing delay of an event and the maximum backlog (work accumulation)

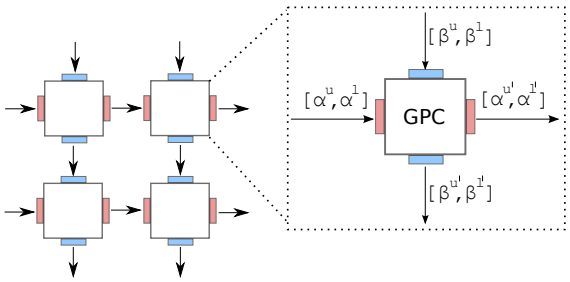

Figure 1: Fragment of a hypothetical process network of GPC's.

can be obtained. For equations and further details see [Wan06]. Figure 1 shows a fragment of a hypothetical process network of connected GPCs.

The described models allow to specify all elements of the system. By interconnecting these elements, an entire system architecture can be modeled and analyzed. As a result, performance metrics, like end-to-end latencies and buffer requirements, can be obtained.

4 System Architecture

Our approach aims at providing an infrastructure for virtualization-based fault tolerant solutions in connection with a distributed management and monitoring system (MMS). We aspire to extend the MMS with a continuous performance analysis mechanism based upon MPA-RTC. The integration of such a mechanism would, at any time, allow the system to be aware of its performance and use the analysis results to automatically reconfigure itself in case of faults. Further, we expect our architecture to be suited to support real-time systems.

The proposed architecture assumes a virtualization technique called native or bare metal. In this approach the hypervisor runs, as a thin software layer, directly on the hardware. This allows a performant concurrent execution of multiple operating systems, without the disadvantages of hosted virtualization. For the prototype implementation of our architecture we have chosen Xen [BDF+03], an established and popular open-source virtual machine monitor (VMM).

The Xen-VMM controls the hardware resources of a system with a technique known as paravirtualization. A privileged domain (called *domain 0*) is created at boot time, holds the actual hardware drivers and is responsible for managing other guests domains (called *guest operating systems*). To issue I/O requests, the unprivileged domains have to use the virtual drivers. The issued calls are then redirected by the hypervisor to domain 0 for execution.

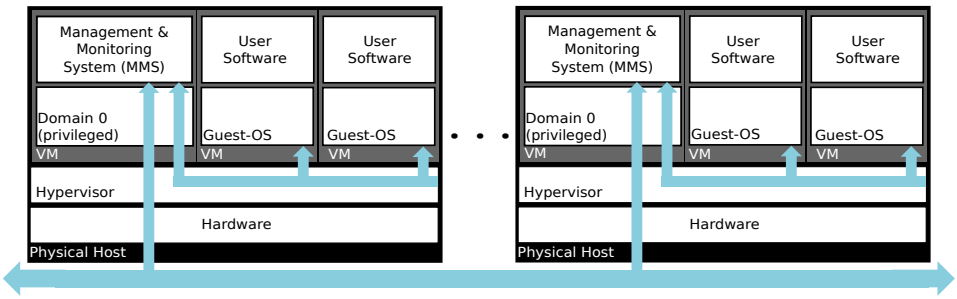

Figure 2: The envisioned system architecture.

Our architecture assumes two communication interfaces. One for the local data exchange within a computing node (a physical host with n-instances of virtual machines) and another for the inter-node communication. Figure 2 shows the envisioned system architecture.

4.1 Management and Monitoring System

Management and monitoring techniques can be implemented in the hypervisor or as privileged components. Our architecture uses the privileged domain to add new management, monitoring and analysis functionality. In the following, a general outline of the MMS units and their functions is given.

4.1.1 Management Unit

In standard mode we assume this unit to be responsible for a smart distribution of workload within a computing node. In case of fault, maintenance- or optimization activities, which exceed the service capacities of a single host, a dynamic inter-node reconfiguration process is to be started. In order to fulfill timing constraints, such an operation has to be preceded with an analysis of possible and system wide load distribution scenarios. This process of gathering *a priori* knowledge about the alternative system configurations and their non-functional characteristics is what we call *continuous performance analysis*. As a result, we obtain a *proactive reconfiguration* mechanism for our architecture. The analysis itself is assumed to be done by the MPA-RTC module, allocated in the monitoring unit. The role of the MPA-RTC module is to provide timing predictions respectively execution latencies for a given computational node with a given workload. Finally, operations like starting, stopping, resuming, migrating and replicating of virtual machines are also located in the management unit.

4.1.2 Monitoring Unit

To enable the envisioned management system, the monitoring unit has to supply all the necessary data. This unit is responsible for the timing and functional examination of virtual machines and the delivering of performance metrics through the MPA-RTC module.

5 Application Scenarios

In the following section, we present two scenarios for our architecture. The first scenario illustrates the modeling abstraction level and the concept of three different execution modes. The second shows a hypothetical use case and demonstrates the application of the MPA-RTC module. In both scenarios we use preemptive fixed-priority scheduling policy to schedule computation and communication resources.

5.1 Scenario 1

At the beginning of our research, to facilitate analysis and evaluation, we chose the following modeling abstraction level of the underlying systems: The GPCs are processing virtual machines with only one guest operating system and one application. The resource nodes represent the server communication capacity respectively the computation capacity with the scheduling policy of the installed hypervisor (in this case FP). In the future, we plan to extend our analysis and allow GPCs to host virtual machines with multiple applications running.

Our architecture assumes the concurrent execution of virtual machines in different modes. The execution modes, in which an application, respectively virtual machine, is running, depend on the priority level of the application. Through different execution modes we plan to realize a variety of fault-tolerant mechanisms. We plan to support three execution modes:

1. normal execution
2. redundant execution on a single host
3. redundant execution or backup on multiple hosts

Non-critical applications with low or no safety implications are supposed to run in normal mode. Normal mode means that the application is being executed as a single instance with no redundancy. This mode has the lowest resource costs, but offers only the standard fault-tolerance mechanisms of a virtualization-based approach. The second mode was designed with the intention to model applications which additionally require parallel execution due to safety or security reasons. This mode provides local redundant execution on a single physical host and is thought to support fault-tolerance techniques like N-version

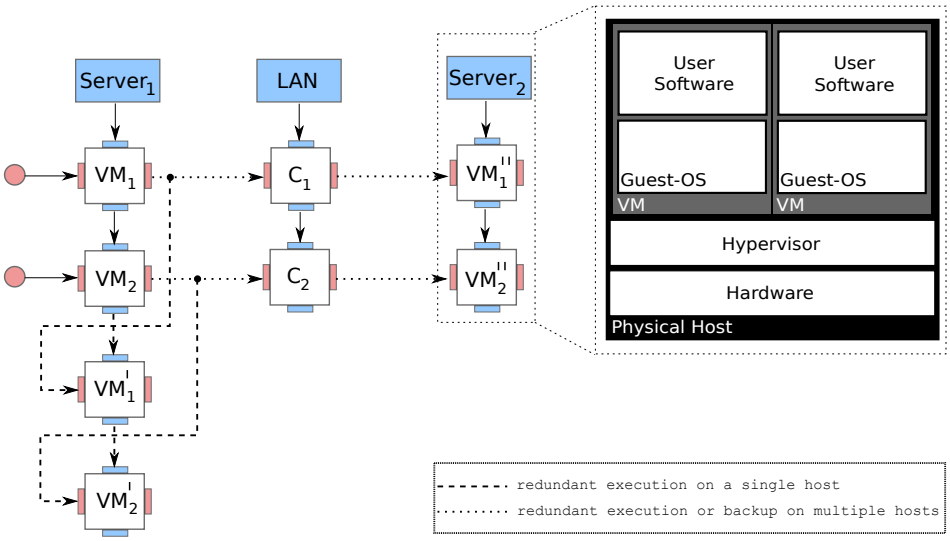

Figure 3: Modeling abstraction level and different execution modes.

programming. The last mode realizes redundant execution on multiple hosts and was designed with the idea to model critical applications, which have to overcome hardware faults. It is also possible to combine mode two with three (mode one is implicitly involved). Such an combination would make an application more robust and resilient, but at the cost of resources. The level of costs has to be determined in further studies.
Figure 3 depicts the first scenario, which illustrates the modeling level and the concept of the three execution modes. The servers are hosting virtual machines which comprise of an application and an underlying and tailored operating system. The two virtual machines (VM_1 and VM_2) are running in all three modes. $VM_1^{'}$ and $VM_2^{'}$ represent the redundant execution on a single host and $VM_1^{''}$ and $VM_2^{''}$ denote the one on different physical hosts. In how many modes an application is to be executed depends on the availability and dependability requirements for the given application.

5.2 Scenario 2

As described in section 3, MPA-RTC can analyze systems with distributed resources shared by independent and interconnected applications. The following scenario shows, how we plan to exploit this property in our architecture. In this simple example our underlying system consists of three computing and one communication resource. There are five tasks running on five GPCs. VM_1, VM_2 and VM_3 are processing three independent event streams. $VM_2^{''}$ is the backup system for the critical task VM_2. C_1 represents the communication task in a local area network (LAN). All the processed events are periodic with the following periods: $a_1 = 25, a_2 = 20, a_3 = 10$. For safety reasons, all deadline

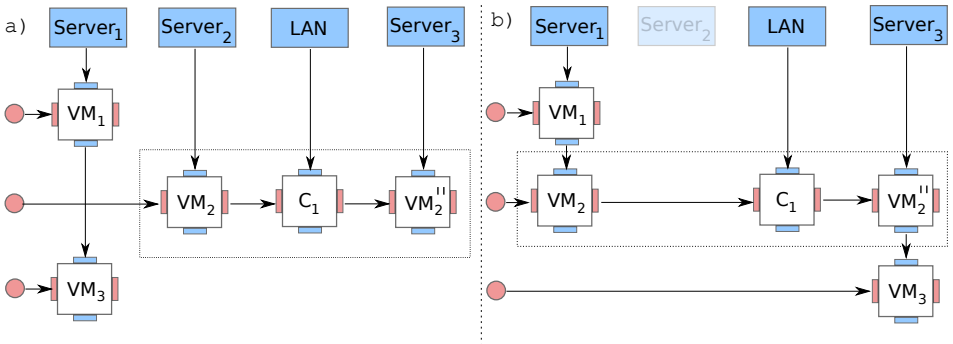

Figure 4: a) standard system configuration. b) optimal system configuration after failure of server$_2$.

have to be guaranteed. Let us now specify the computation/communication demands and deadlines of tasks. We use the following notation: $task = ([WCET, BCET], deadline)$. Then we have: $VM_1 = ([3, 2], d = 4), VM_2 = ([5, 2], d = 10), VM_3 = ([2, 1], d = 6), VM_2^{''} = ([2, 1], d = 2)$ and $C_1 = ([1, 1], 2)$. Further, for the sake of clarity, we normalize the service of server$_1$, server$_2$, server$_3$ and LAN to one unit per unit time interval. Now we can use the MPA-RTC module to calculate the latencies for the processed event streams and verify, if all the deadlines were met. The worst-case delays of the processed events are as follows: $a1 = 3, a2 = 8, a3 = 5$. So, in the considered case, the system is working properly and no deadlines were missed. We intend to use such a standard, no-failure system state to perform proactive reconfiguration. The MMS module would permute the possible configuration options of the given system, analyze them with respect to performance metrics and save the optimal configuration. Figure 4 a) depicts the starting configuration of the examined systems scenario.

In the example being considered, suppose the failure of server$_2$. Because of the implemented proactive reconfiguration technique we assume our architecture, to be prepared, to handle such a situation without the need of extra computation time. Figure 4 b) depicts the new, optimal system after MMS transformation in case of the considered fault. There were other possible configuration options. VM_2 could be migrated to server$_3$, however this would cancel the fault-tolerance mechanism against hardware failure. In turn, only migrating VM_2 to server$_2$ without migrating VM_3, would cause VM_3 to miss its deadline. For the sake of understanding, this example was intentionally kept very simple. In a realistic system with many applications, real-time scheduling and communication challenges, hardware restrictions and complex arrival patterns for events with jitter, a formal performance analysis technique is vital and indispensable to enable our envisioned mechanism for proactive reconfiguration.

6 Discussion

In this section, we shortly discuss some of the challenges that our novel approach rises. As mentioned in section 3, to apply the performance analysis with MPA-RTC, prerequisite information have to be determined. How well the the worst/best-case behavior of virtual machines or the timing properties of the hypervisor can be approximated is to be investigated. Further, not all the information is always available and some of the systems can't be completely, statically planned. In a dynamically changing, technical context some of the events, or the workload they impose on a system, can't be determined with certainty. Next, the modeling scope of our approach is restricted to that of MPA-RTC. This complicates the analysis of state-based components. Another issue is the runtime of our MMS application. Even with proactive reconfiguration, automatically generated system models and evaluation scripts for the MPA-RTC module, in case of real-time constraints, running times are to be very efficient. Also the execution times, service downtimes, performance degradations of primary VMs and communication overheads are to be investigated. Finally, there must be an evaluation if the savings through system-integration outweigh the costs incurred by high availability solutions.

7 Conclusions and Future Work

The combination of virtualization-based fault-tolerant systems with real-time constraints yields many challenges. In this paper, we have presented a novel architecture, which aims at overcoming these challenges, by incorporating recent fault-tolerance mechanisms, real-time scheduling and proactive reconfiguration based upon continuous performance analysis.

In the nearest future, we intend to evaluate the functional and non-functional properties of our approach in detail. The designed architecture will be implemented in Xen and recent high availability and scheduling extensions will be incorporated. For the purpose of validation, the MPA-RTC will be embedded. We believe that our envisioned, virtualization-based architecture is suited to satisfy the sophisticated requirements of many modern real-time systems that depend upon fault-tolerance.

References

[AC77] A. Avižienis and L. Chen. On the implementation of N-version programming for software fault tolerance during execution. In *Proceedings of the IEEE International Computer Software and Applications Conference*, pages 149–155, 1977.

[BCOQ92] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. Synchronization and linearity : An algebra for discrete event systems. Wiley Series in probability and mathematical statistics, 1992.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[BS96] Thomas C. Bressoud and Fred B. Schneider. Hypervisor-based fault tolerance. *ACM Trans. Comput. Syst.*, 14(1):80–107, February 1996.

[BTS+10] Felix Bruns, Shadi Traboulsi, David Szczesny, Elizabeth Gonzalez, Yang Xu, and Attila Bilgic. An Evaluation of Microkernel-Based Virtualization for Embedded Real-Time Systems. In *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, ECRTS '10, pages 57–65, Washington, DC, USA, 2010. IEEE Computer Society.

[CKT03] Samarjit Chakraborty, Simon Kunzli, and Lothar Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 10190–, Washington, DC, USA, 2003. IEEE Computer Society.

[CLM+08] Brendan Cully, Geoffrey Lefebvre, Dutch T. Meyer, Mike Feeley, Norman C. Hutchinson, and Andrew Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. (Best Paper). In Jon Crowcroft and Michael Dahlin, editors, *5th USENIX Symposium on Networked Systems Design & Implementation, NSDI 2008, April 16-18, 2008, San Francisco, CA, USA, Proceedings*, page 161. USENIX Association, 2008.

[Cru91] Rene L. Cruz. A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[HHB+12] Kai Huang, Wolfgang Haid, Iuliana Bacivarov, Matthias Keller, and Lothar Thiele. Embedding Formal Performance Analysis into the Design Cycle of MPSoCs for Real-time Streaming Applications. *ACM Transactions in Embedded Computing Systems (TECS)*, 11(1):8:1–8:23, 2012.

[KDVW97] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter van der Wolf. An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, ASAP '97, pages 338–, Washington, DC, USA, 1997. IEEE Computer Society.

[LT01] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[LXC+12] Jaewoo Lee, Sisu Xi, Sanjian Chen, Linh T.X. Phan, Christopher Gill, Insup Lee, Chenyang Lu, and Oleg Sokolsky. Realizing Compositional Scheduling through Virtualization. In *In Proceedings of the IEEE Symposium on Reliable Distributed Systems*, 2012.

[Per11] Simon Perathoner. *Modular performance analysis of embedded real-time systems: improving modeling scope and accuracy*. PhD thesis, 2011.

[RHKSP06] Hans P. Reiser, Franz J. Hauck, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Hypervisor-Based Redundant Execution on a Single Physical Host. In *Proceedings of the 6th European Dependable Computing Conference, Supplemental Volume - EDCC'06 (Oct 18-20, 2006, Coimbra, Portugal)*, pages 67–68, 2006.

[RK07] Hans P. Reiser and Rüdiger Kapitza. Hypervisor-based efficient proactive recovery. In *In Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pages 83–92, 2007.

[TKSS08] Yoshi Tamura, S. Koji, K. Seiji, and M Satoshi. Kemari: Virtual Machine Synchronization for Fault Tolerance, 2008.

[Wan06] Ernesto Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, PhD Thesis ETH Zurich, 2006.

[WTVL06] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System Architecture Evaluation Using Modular Performance Analysis - A Case Study. *Software Tools for Technology Transfer (STTT)*, 8(6):649 – 667, 2006.

[XWLG11] Sisu Xi, Justin Wilson, Chenyang Lu, and Christopher Gill. RT-Xen: towards real-time hypervisor scheduling in xen. In *Proceedings of the ninth ACM international conference on Embedded software*, EMSOFT '11, pages 39–48, New York, NY, USA, 2011. ACM.

[ZDJ+10] Jun Zhu, Wei Dong, Zhefu Jiang, Xiaogang Shi, Zhen Xiao, and Xiaoming Li. Improving the performance of hypervisor-based fault tolerance. In *IPDPS*, pages 1–10. IEEE, 2010.