A First Step towards Real-Time Virtual Machine Migration in Heterogeneous Multi-Processor Systems

S. Groesbrink

Design of Distributed Embedded Systems, Heinz Nixdorf Institute, University of Paderborn, Germany
E-Mail: stefan.groesbrink@hni.upb.de

Summary: The combination of system virtualization and heterogeneous multi-processor architectures can help to create efficient platforms that meet the requirements of increasingly complex embedded systems. Virtualization adds runtime flexibility and facilitates by modular synthesis the creation of a system of systems. However, existing virtualization solutions are characterized by a completely static mapping of virtual machines to processors and do not use the full potential of heterogeneous architectures. The presented solution applies migration and emulation to realize a dynamic assignment of virtual machines with real-time constraints. It identifies the necessary conditions for migration and the degree of communication between hypervisor and operating system that is indispensable for migration decisions.

Keywords: System Architecture, Algorithm, Flexibility, Reliability.

1. Introduction

There is a trend towards adaptability and self-optimization for advanced mechatronic systems [1]. As a reaction to defects of components or changes of the environment, self-optimizing mechatronic systems adjust their behavior at runtime. The realization of such systems demands dynamic system software architectures, since the traditional static approaches for embedded software are inappropriate for the varying resource requirements of adaptive systems. The primary objective for mechatronic systems remains the guarantee of real-time behavior.

1.1. System Virtualization

System virtualization is a powerful technology for integrated adaptive systems. State-of-the-art in the enterprise and server space for a fairly long time, virtual machine technologies are now gaining acceptance in the field of embedded systems [2]. A virtualization layer called virtual machine monitor or hypervisor is added between hardware (CPU, memory, I/O devices) and operating system (OS). This allows for the concurrent execution of multiple OSs: each guest OS is executed in a dedicated virtual machine (VM) and gets the illusion of accessing the real machine exclusively (Figure 1). The underlying hardware can be shared and utilized effectively with maintained isolation between the subsystems, which is important for critical mechatronic systems.

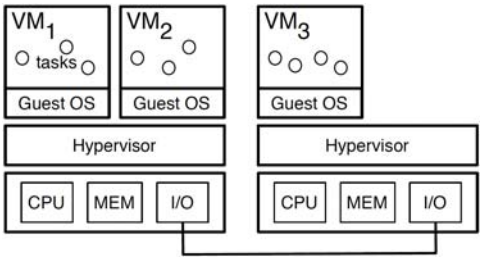

Figure 1. System Virtualization in a Multi-processor Architecture.

1.2. Heterogeneous Multi-Processor Systems

Multi-processor systems are already very common in the field of mechatronic systems and have the potential to provide powerful and efficient platforms that meet the high functionality requirements of upcoming products. Multi-processor technology is a major enabler for virtualization, since the architectural abstraction aspect of virtualization allows an essentially unmodified porting of single-core implementations to multi-processors without re-architecting [3]. In contrast to the shared random access memory (RAM) of multi-*core* architectures, the memory is distributed in a multi-*processor* system.

Processor heterogeneity is a closely related trend. The different requirements of the subsystems can be met with adequate processors. Heterogeneous platforms enable the consolidation of legacy systems that were developed for different processor architectures. A popular example of a heterogeneous multi-processor architecture is the Xilinx Virtex series of FPGAs (Figure 2). It combines a highly configurable MicroBlaze processor and a PowerPC processor for performance oriented applications [4]. The processors work on their own RAM, but are connected by a bus hierarchy and can use shared memory, which is connected to the bus as well.

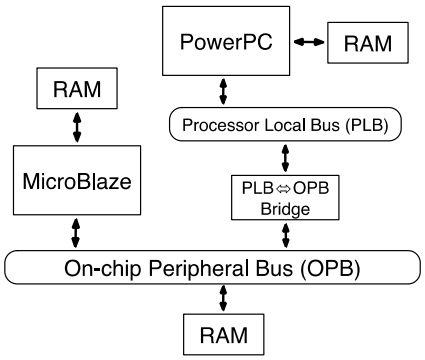

Figure 2. Example for a Heterogeneous Multi-processor System (cf. [4]).

1.3. Virtual Machine Migration

Migration refers to the relocation of virtual machines from one processor to another one at runtime. Prerequisite is a platform of multiple connected processors, each of them running a hypervisor (Figure 1). Migration offers several advantages. It enables load balancing and increases the robustness by fault resilience in case of partially failed processors. The increased flexibility can be exploited to facilitate adaptability and optimize performance and resource utilization.

Emulation refers to the implementation of the interface and functionality of one system on a system having a different interface and functionality. With the use of this technique, virtualization enables migration even in a heterogeneous processor environment. Emulation realizes cross-platform software portability by supporting program binaries compiled for an instruction set architecture (ISA) that is different from the ISA of the processor on which it is executed.

1.4. Motivation

Virtualization solutions for the server market apply highly dynamical approaches, are however not real-time capable, a fundamental requirement for mechatronic systems. In order to guarantee real-time requirements, existing virtualization solutions for embedded systems typically assign the virtual machines statically to the processors. This is not appropriate for highly dynamic adaptive systems. This paper presents a more flexible virtualization solution for intelligent mechatronic systems, based on migration and emulation.

2. Virtual Machine Migration for Real-time Systems

A real-time constraint guaranteeing migration has to assure that the resource requirements of all virtual machines can still be met after the migration. In addition, it has to be guaranteed that the next deadline is not endangered by the interruption time resulting from the migration process (downtime). This is accomplished by the presented approach, which is based on the basic architecture of Figure 3. The individual components are explained in the following.

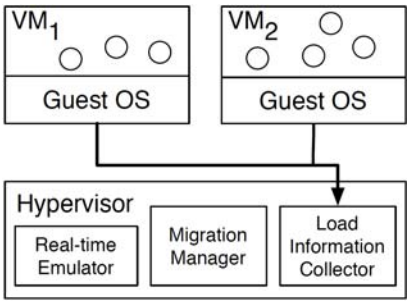

Figure 3. Main components of the virtual machine migration approach.

2.1. Real-time Emulator

An emulation component as part of a hypervisor for mechatronic systems has to guarantee real-time response times and a completely predictable timing behavior. In previous work in the context of this project, Kerstan and Oertel developed an emulation method, which combines the two basic emulation approaches interpretation and binary translation for an optimal trade-off between required memory and performance [5]. Of major importance, the approach derives the processor speed that is required to meet all deadlines. For each possibility to execute software that was compiled for ISA_{host} and that can be emulated on ISA_{target}, the instruction of ISA_{host} that is slowed down the most on the target processor is crucial. It determines the required speed for the target processor to meet the timing constraints, more precisely the speed-up factor, which denotes how much faster the target processor has to be in comparison to the host processor:

$$S_{host,target} = \max\left(\left\{\frac{\#TargetCycles(emu_{host2target}(instr))}{\#SourceCycles(instr)}\right\} \middle| instr \in ISA_{host}\right)$$

The calculation is based on the ratio of the execution times in number of clock cycles. $emu_{host2target}(instr)$ determines the sequence of instructions, which has to be executed to emulate the instruction *instr*.

The calculation of the speed-up factor is for multiple reasons a pessimistic overestimation. First, it is assumed that the cycles per instruction are constant, not considering dynamic optimizations of the processor. Second, a program may not call the instruction that is slowed down the most, but this instruction is nevertheless included in the determination of the required speed-up. These overestimations allow for a static computation of the speed-up factor, based only on the characteristics of the processors and independent from the actual executed software, which is of great help for open and adaptive systems.

2.2. Load Information Collector

The component for load information collection gathers at runtime data about the resource utilization. The analysis of this data is the base for the migration decisions taken by the migration manager. The hypervisor assigns the resources to the VMs and has therefore knowledge about the guest's memory and I/O usage. However, the hypervisor does not have any insight in the scheduling of the guest operating system and does by consequence not know the next deadline. Therefore, it cannot evaluate whether a certain downtime invokes a deadline miss or not.

Consequently, an explicit communication between guest OS and hypervisor is required. Whenever the hypervisor considers a migration, it invokes the OS to pass information about the next deadline of its tasks. The communication of the continually changing next deadline would imply an inappropriate overhead and therefore this information is only communicated upon request.

2.3. Migration Manager

The migration manager is the decision-making component of the migration process and calls the emulator and the collector. It decides *which* virtual machine shall be migrated *where* and *when*. These decisions are based on the data about resource utilization and deadlines, provided by the load information collector. In the following, we will look in detail on the evaluation of possible migrations. We assume that the hypervisor includes emulation functionality for all processor architectures of the platform, which implies that all virtual machines can be executed on all processors.

There are two necessary conditions for a migration:

1. The resource utilization of the target processor must be low enough to permit the addition of the virtual machine (CPU, memory, I/O). This condition is checked by the *acceptance test*.
2. The downtime imposed by the migration process must be short enough to exclude a miss of the next deadline of the guest system. This condition is checked by the *migration test*.

Before deciding pro migration, the migration manager has therefore to conduct the following two tests:

2.3.1. Acceptance Test

The test is based on the question whether the remaining resources are sufficient to fulfil the resource requirements of the arriving VM. Regarding memory, a simple comparison of unassigned memory and demanded memory is needed. Regarding I/O resources, we assume that the I/O devices are accessible uniformly by all processors and no check has to be performed for these components in the acceptance test.

The test regarding the resource computation time is much more complex. If multiple real-time VMs are assigned to one processor, only one VM can be executed at each point in time. The hypervisor decides which VM is active and all other VMs experience a blackout. A guarantee that all deadlines are met cannot be given based solely on the CPU utilization, since periodicity and length of the activation slots have a major impact.

Dynamic solutions that do not require knowledge about all executed tasks are unknown. For this reason, our first approach assigns each hard real-time VM to a dedicated processor. An addition of non-real-time VMs to a processor that hosts already a real-time VM is however possible. In this case, the non-real-time VMs are executed whenever the real-time VM is not running. If a passed memory requirement test is assumed, a non-real-time VM is always allowed to migrate.

If a real-time VM shall be migrated, the request can be accepted if the target processor does not host already a real-time VM. This condition is however not sufficient, since in a heterogeneous multi-processor architecture, the execution time of an application depends on the processor. If only the processor speed differs, the migration request can be accepted if the speed of the target processor is equal or greater than the speed of the host processor. If the instruction set architecture differs, the overhead introduced by emulation has to be added. As a consequence of this overhead, the target processor must have a speed of at least $S_{host,target}$, as introduced in section *2.1*.

2.3.2. Migration Test

The worst-case virtual machine downtime must be predictable for a real-time system. Only under this condition, it is possible to evaluate whether the downtime leads to a missed deadline or not. The downtime adds up by the following time intervals:

1. Detaching. The execution of the VM is suspended and a buffering of the communication is set up. The executable code, memory, and processor state (content of registers, set flags) are wrapped up to a transferrable data package.
2. Transfer. The extracted VM state is communicated to the target processor.
3. Resume. The VM state is unwrapped, the code loaded, the memory copied, the processor state reset, and the buffered communication processed.

The migration test is passed, if the remaining time until the next deadline is equal or greater than the downtime plus the time required until the remaining execution time of the task is completed on the target processor.

2.3.3. Migration Decision

The last two sections dealt with the conditions that are necessary for migration. However, it is of course not intended to migrate whenever possible. In fact, a reasonable trade-off between adaptability and stability has to be found. It is in general plausible that migration should be performed if the functioning of a subsystem can continue despite a hardware failure. Very similar, migration is very valuable for open systems, since it can allow for the acceptance of an arriving subsystem in situations, in which this is not possible without migration. Whether a migration should be conducted to improve the load balancing is much more application-specific and it is difficult to define general yields.

3. Conclusion

This paper presents an approach for the emulation-based migration of virtual machines with real-time constraints in heterogeneous multi-processor systems. The communication between hypervisor and guest operating system was examined and the indispensable information passing to decide pro migration without endangering real-time constraints was identified.

The approach is currently integrated into our real-time hypervisor Proteus [6]. The implementation requires paravirtualization, that is to say a modification of the guest operating systems. This necessity results from the fact that the guest operating systems have to pass information to the hypervisor.

Acknowledgements

This work was supported by the German Collaborative Research Center 614 - Self-Optimizing Concepts and Structures in Mechanical Engineering (SFB614, www.sfb614.de).

References

[1] Pook, S., Gausemeier, J., Dorociak, R., 2012, Securing the Reliability of Tomorrow's Systems with Self-Optimization, The Annual Reliability and Maintainability Symposium.

[2] Heiser, G., 2008, The Role of Virtualization in Embedded Systems, The 1st Workshop on Isolation and Integration in Embedded Systems.

[3] Neumann, D., 2006, Intel Virtualization Technology in Embedded and Communication Infrastructure Applications, Intel Technology Journal, 10/3.

[4] Xilinx, Inc., 2002, Virtex-II Pro Platform FPGA Handbook.

[5] Kerstan, T., Oertel, M., 2010, Design of a Real-time Optimized Emulation Method, Proceedings of the Conference on Design, Automation, and Test in Europe.

[6] Baldin, D., Kerstan, T., 2009, Proteus, a Hybrid Virtualization Platform for Embedded Systems, Proceedings of the 3rd International Embedded Systems Symposium.