Fair Bandwidth Sharing among Virtual Machines in a Multi-criticality Scope

Stefan Groesbrink
Heinz Nixdorf Institute
University of Paderborn
Paderborn, Germany
s.groesbrink@upb.de

Luis Almeida, Mario de Sousa
IT - Faculty of Engineering
University of Porto
Porto, Portugal
{lda, msousa}@fe.up.pt

Stefan M. Petters
CISTER/INESC-TEC,
ISEP, IPP
Porto, Portugal
smp@isep.ipp.pt

***Abstract*—System virtualization's consolidation in separated virtual machines provides a reasonable way to integrate formerly distinct systems into a single mixed-criticality multi-core system. We propose an adaptive resource management scheme for virtualization-based systems that have to be certified. Periodic servers and the elastic task model combine analyzability at design time with adaptability at runtime. A mode change or the enabling / disabling of tasks trigger a resource redistribution, which guarantees that a specified minimum is always allocated and obtains a fair distribution of spare capacity among the virtual machines. The partitioned scheduling and the assignment of static priorities ease certification. The scheme has the potential to improve the resource utilization and support adaptive and self-optimizing applications with strongly varying execution times.**

I. Introduction

System virtualization refers to the hypervisor-controlled division of the resources of a computer system into multiple virtual machines (VM). Each VM runs an adequate operating system, e.g., an efficient and highly predictive real-time executive for safety-critical control tasks and a feature-rich operating system for the communication or human-machine interface. The consolidation of multiple systems with maintained isolation and resource partitioning is well-suited to combine independently developed trusted (and potentially certified) systems of different criticality levels to a system-of-systems. The rise of multi-core embedded processors [1] is a major enabler for virtualization, whose architectural abstraction eases the migration from single-core to multi-core platforms. The replacement of multiple hardware units by a single multi-core system has the potential to provide the required computational capacity with reduced size, weight, and power.

In order to guarantee real-time requirements, existing virtualization solutions for embedded systems apply a static resource assignment [2]. Two typical solutions are an one-to-one mapping of VMs to processors, or if VMs share a processor, a static cyclic schedule with fixed execution time slices. Static resource allocation naturally evokes fragmentation of available resources, as reserved, but unused capacity cannot be reclaimed to improve the performance of other VMs. Static approaches are in addition inappropriate for the varying resource requirements of adaptive and self-optimizing systems as well as open systems, in which subsystems may be added or removed at runtime.

In this work, we propose an adaptive resource management scheme based on a dynamic budget setting of servers combined with an elastic task model. Next to the appropriate consideration of multiple criticality levels, one goal is the fair distribution of spare capacity among the virtual machines. This is particularly important for virtual machines that either run applications with strongly varying execution time or that can take advantage of higher capacity to produce improved results or provide extra functionality. This is a work-in-progress paper and the necessary validation is still on-going.

II. System Model

The periodic task model [3] defines workloads as a sequence of jobs. Each task τ_i is characterized by a worst-case execution time (WCET) C_i and a period T_i, denoting the time interval between the activation times of consecutive jobs. The utilization of a task $U(\tau_i)$ is defined as $U(\tau_i) = C_i/T_i$. Moreover, a criticality level χ is assigned to each task [4], lower value denoting higher criticality. The presented allocation scheme can be applied to the sporadic task model as well, but the periodic model is more suitable for certification.

A VM V_k is modeled as a set of tasks and a guest operating system. Notation $\tau_i \in V_k$ denotes that task τ_i is executed in V_k. The utilization of V_k is the sum of the utilizations of its tasks, implying that operating system overhead is neglected: $U(V_k) = \sum_{\tau_i \in V_k} U(\tau_i)$. The criticality level of V_k corresponds to the highest criticality of its tasks: $\chi(V_k) = max\{\chi(\tau_i) | \tau_i \in V_k\}$. Independent VMs are assumed, with neither shared resources except from the processor, nor data dependencies, nor inter-VM communication. $V_k \mapsto Proc_l$ denotes that VM V_k is executed on processor $Proc_l$.

Marau et al. extended the periodic task model in order to realize an *elastic scheduling* [5]. A task τ_i is characterized by a minimum bandwidth $U_{min}(\tau_i)$, a maximum bandwidth $U_{min}(\tau_i)+U_{lax}(\tau_i)$, and a Quality-of-Service (QoS) parameter $qos(\tau_i)$. Given a (fixed) resource capacity U_R, the spare bandwidth defines the bandwidth that can be distributed among tasks after having guaranteed their minimum requirements: $U_{spare} = U_R - \sum_{i=1}^{n} U_{min}(\tau_i)$. Marau et al. presented an algorithm that provides a weighted distribution among the services, in order to achieve a notion of fairness regarding the QoS improvement.

III. ARCHITECTURE

A. Abstraction of Resource Supply by Virtual Processors

Target platform are homogeneous multi-core systems, consisting of m identical cores. A virtual processor P_k^{virt} is a representation of a share of the bandwidth of a physical processor to the VM V_k and multiple virtual processors can be mapped to a single physical processor. A formal abstraction of the resource supply by a virtual processor was introduced by Shin and Lee [6]. The *periodic resource model* $\Gamma(\Pi, \Theta)$ characterizes a periodic behavior of a partitioned resource: Θ time units are allocated every period Π ($0 < \Theta \leq \Pi$). The allocation according to the periodic resource model is a natural fit to the periodic timing requirements of the applied workload model. If a processor is not partitioned, an exclusively assigned processor is modeled as $\Gamma(1, 1)$. The minimum computation time allocations provided by a virtual processor in a time interval of length t are specified in terms of a supply bound function sbf(t), which allows to analyze the schedulability of a VM as if it is executed by a fractional processor.

Each virtual processor $\Gamma_k(\Pi_k, \Theta_k)$ is implemented as a periodic server. If scheduled and therefore active, a server's capacity is used to execute the computation time demand of the associated VM. The capacity limits this service and is replenished at each period. The server enforces a certain bandwidth α, limiting the contribution of a VM, even in the presence of overloads:

$$\alpha_k = \frac{\Theta_k}{\Pi_k} \tag{1}$$

Moreover, the service delay Δ specifies the maximum period of time that the associated VM may have to wait before receiving computational service [7]:

$$\Delta_k = 2 \cdot (\Pi_k - \Theta_k) \tag{2}$$

B. Partitioned Hierarchical Scheduling

System virtualization implies partitioned scheduling, since entire software stacks including operating system are integrated, resulting in scheduling decisions on two levels (hierarchical scheduling). The hypervisor schedules the VMs 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. To enable the hypervisor-based integration of independently developed and validated systems, temporal isolation between VMs has to be ensured by a hierarchical scheduling scheme and there are many solutions for uniprocessor hierarchical real-time scheduling, e.g. [8], [9].

1) Virtual Machine Scheduling: In the context of this work, VMs are statically assigned to 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.

The partitioning of the VMs to the processors focuses on two goals. Minimizing the overall required computation bandwidth is the first goal, since it determines the number of processors required to host the set of VMs. In addition, a distribution of critical VMs among the processors is targeted. If VMs of differing criticality share a processor, there are in general more possibilities to apply an adaptive bandwidth management and the addition of subsystems of low criticality to critical subsystems can increase the resource utilization, as higher criticality applications suffer from a more pessimistic worst-case resource demand determination [10].

A dedicated virtual processor—modeled as a periodic resource model $\Gamma_k(\Pi_k, \Theta_k)$ and realized as a periodic server—is assigned to each VM V_k. The initial bandwidth of a server is determined by the minimum bandwidth $U_{min}(V_k)$. A small period causes a high number of costly virtual machine context switches and a large period results in a large service delay and hence in a low reactivity. The largest possible *blackout* interval without resource supply determines the largest possible service delay Δ_k^{max}. A necessary condition is that Δ_k^{max} is not greater than the smallest difference of period and WCET for all tasks of VM_k:

$$\Delta_k^{max} \leq \min_{\tau_j \in V_k} (T_j - C_j) \tag{3}$$

With the largest possible service delay Δ_k^{max} known for each VM V_k and Equation 3 and Equation 4 given, the server parameters for the virtual processor assigned to V_k are determined as follows:

$$\Pi_k = \frac{\Delta_k^{max}}{2 \cdot (1 - \alpha_k)} = \frac{\Delta_k^{max}}{2 \cdot (1 - U_{min}(V_k))} \tag{4}$$

$$\Theta_k = \Pi_k \cdot \alpha_k = \Pi_k \cdot U_{min}(V_k) \tag{5}$$

According to the scheme *Criticality as Priority Assignment* (CAPA) [11], static priorities are assigned to the servers in the order of decreasing criticality level. Starvation of lower criticality VMs is nevertheless precluded, since a server is suspended when the capacity is exhausted, unless no other VM is active or, if it is a soft server, moved to execute in a lower priority band.

2) Task Scheduling: A guest operating system can apply an arbitrary task scheduling algorithm *A*, as long as it allows to abstract the computation time requirements of the VM for a specific time interval of length t in terms of a *demand-bound function* $dbf_{A,V_i}(t)$ [6]. The comparison of bounded demand of a VM and bounded supply by the associated virtual processor realizes the schedulability analysis.

C. Dynamic Bandwidth Distribution

The adaptive resource management is implemented by a dynamic modification of the budgets of the servers. A minimum bandwidth is guaranteed for each VM and additional bandwidth can be assigned. The minimum bandwidth $U_{min}(V_k)$ of the server associated to VM V_k is set to the sum of the minimum bandwidths of V_k's tasks:

$$U_{min}(V_k) = \sum_{\tau_i \in V_k} U_{min}(\tau_i) \tag{6}$$

Since VMs are statically assigned to the processors, spare bandwidth has to be handled separately for each processor. $U_{spare}(l)$ specifies the spare bandwidth of processor $Proc_l$.

According to the elastic task model, the spare bandwidth of $Proc_l$ is equal to:

$$1 \leq l \leq m : U_{spare}(l) = U_R - \sum_{V_i \mapsto Proc_l} U_{min}(V_k) \quad (7)$$

The server period Π_k is fixed, whereas the capacity Θ_k is set dynamically in order to allocate bandwidth in an adaptive manner. The distribution algorithm considers the two factors criticality level and weight in order to determine which VM receives how much spare bandwidth. The criticality level χ is the dominant factor and in a first step, the bandwidth is assigned in a greedy manner in order of decreasing criticality. The highest criticality level obtains as much bandwidth as possible, limited by either the distributable amount U_{spare} or the maximum bandwidth of its VMs:

$$\sum_{V_i \text{ with } \chi(V_i)=max\_crit} (U_{min}(V_i) + U_{lax}(V_i)) \quad (8)$$

If there is spare bandwidth left, the next lower criticality level is served and so on. The weights influence the bandwidth assignment among VMs of the same criticality level, since a greedy strategy lacks fairness. The determination of the weight of a VM is based on the weight of its tasks. The normalized weight of a task τ_j in turn is based on the $qos(\tau_j)$ value [5]:

$$w(\tau_j) = \frac{qos(\tau_j)}{\sum_{\tau_k \in V_i} qos(\tau_k)} \quad (9)$$

$$w(V_i) = \sum_{\tau_j \in V_i} \frac{qos(\tau_j)}{\sum_{\forall \tau_l} qos(\tau_l)} \quad (10)$$

Assumed that the bandwidth U_{spare} is to be distributed among VMs of same criticality level, the shares are set to:

$$U_{add}(V_i) = w(V_i) \cdot U_{spare} \quad (11)$$

This results in a total bandwidth assignment of:

$$U(V_i) = U_{min}(V_i) + U_{add}(V_i) \quad (12)$$

The new server bandwidth its set to $\alpha_i = U(V_i)$ and the replenished budget follows as $\Theta_i = \alpha_i \cdot \Pi_i$.

The goal of the adaptive resource management is maximizing the resource allocation in order of decreasing criticality. Moreover, another goal is to maximize the QoS for the fixed amount of computation capacity, if there are guest systems that can take advantage of an additional allocation of bandwidth.

The hypervisor is in charge of setting the budgets of the servers. The following events have a significant impact on the spare bandwidth U_{spare} and trigger a redistribution:

- mode change of a task / virtual machine
- enabling / disabling of a task / virtual machine

Only $U_{add}(V)$ is determined at runtime and added to the constant $U_{min}(V)$. Therefore, it is precluded that the allocated bandwidth falls below $U_{min}(V)$.

TABLE I: Example: Virtual Machine Set

VM	$modes$	χ	U_{min}	U_{lax}	w
V_1	a	1	.1	0	–
	b	1	.1	.1	–
V_2		1	.1	0	–
V_3		2	.2	.3	.2
V_4		2	.2	.6	.8
V_5		2	.3	.7	–
V_6		3	0	.1	–

IV. Example

The following example illustrates the benefits of a dynamic distribution of spare bandwidth. The set of VMs is given in Table I. The motivating example is the consolidation of safety-critical control systems with QoS-driven computer visions systems for the automotive domain. The VMs V_1 and V_2 host control systems of highest criticality level. Additional resources are of no benefit for some applications ($U_{lax}(V_2) = 0$), since they do not perform mode changes or enable or disable tasks at runtime. V_1 is assumed to use additional bandwidth to produce better results, realized by a mode change.

The VMs V_3, V_4, and V_5 execute computer vision systems, for example, for the detection of other vehicles or objects for a collision warning system or the detection of lanes for a lane-departure warning system. The QoS is directly related to the number of processed frames per second. The execution time of the vision algorithms varies, dependent on the variety of situations and illumination conditions. Such systems benefit from additional resource allocations and both mode changes and task enabling/disabling can be applied subject to the driving situation. For example, the rear view camera system must only be enabled when the car is reversing.

VM V_6 hosts a non-critical system. No service level guarantee is given ($U_{min}(V_6) = 0$) and the VM is scheduled in background. Such a specification is only reasonable if it is known that there actually will be allocations at runtime in the situation in which this VM has demand. An example is the control of an adjustable driver seat, which has a computation demand only if the car is not moving and in this situation the load of the VMs of higher criticality is low.

This example emphasizes the flexibility of the elastic model to support different criticality levels. Naturally modeling applications with varying demand, it can be used as well for highly-critical applications without varying execution time ($U_{lax} = 0$) and for non-critical applications that are scheduled in background ($U_{min} = 0$).

Assuming a two-processor platform, an appropriate partitioning of this VM set is for example a mapping of $\{V_1, V_5, V_6\}$ to processor $Proc_1$ and of $\{V_2, V_3, V_4\}$ to processor $Proc_2$, resulting in spare bandwidths of $U_{spare}(Proc_1)$=0.6 and $U_{spare}(Proc_2)$=0.5. Figure 1 depicts the VM schedule and the bandwidth allocation (dotted box $\widehat{=}Proc_1$, hatched box $\widehat{=}Proc_2$). For the sake of clarity, this example considers only the VM level, not the task level, and a period of 10 is assumed for all servers.

From t=0 until t=10, the schedule without distribution of spare bandwidth is illustrated. V_1 is assumed to be in mode (a),

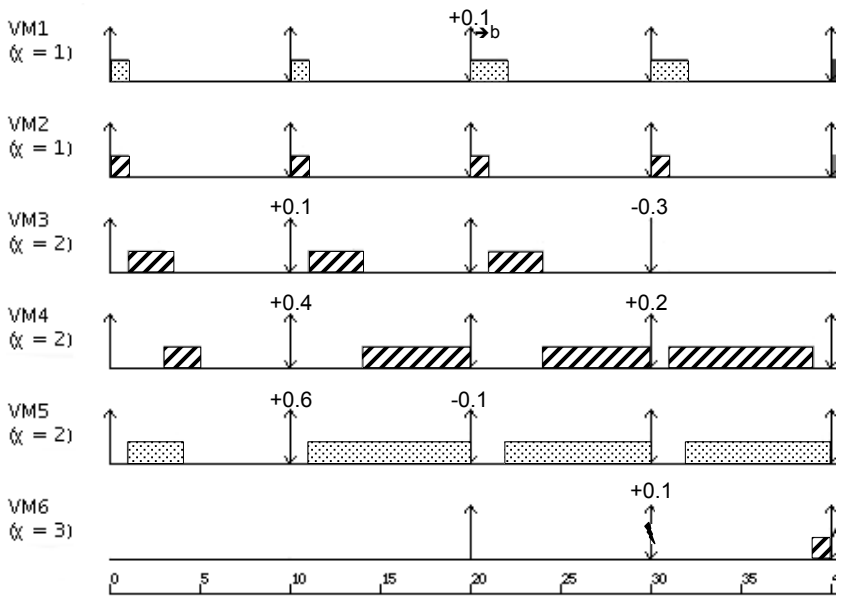

Fig. 1: Example: Virtual Machine Schedule with Adaptive Bandwidth Allocation

V_6 is assumed to be deactivated. At t=10, $U_{spare}(Proc_1)$=0.6 is assigned to V_5, since it is the only guest with a nonzero U_{lax}. On $Proc_2$, V_3 obtains an additional bandwidth of 0.1 and V_4 obtains additional bandwidth of 0.4, according to their weight. At t=20, V_1 switches to mode (b), in which it benefits from extra bandwidth and the bandwidth of the lower-criticality V_5 is reduced accordingly. V_6 becomes active, is however not executed (no guaranteed bandwidth), since the VMs of higher criticality use the entire bandwidth. At t=30, V_3 is deactivated and the released bandwidth is redistributed to V_4 and V_6.

V. Conclusion

We proposed an adaptive resource management scheme for virtualization-based systems that have to be certified. Periodic servers and the elastic task model combine analyzability at design time with adaptability at runtime. Periodic servers provide isolation among VMs and the dynamic budget replenishment implements an efficient bandwidth reallocation. A mode change or the enabling / disabling of tasks trigger a resource redistribution, which handles multiple criticality levels appropriately and aims at achieving a fair distribution of spare capacity among VMs.

The partitioned scheduling and the assignment of static priorities ease certification significantly. The server-based limitation of the computation time of highly-critical applications precludes a starvation of applications of lower criticality. Most important for certification, the bandwidth allocation never falls below the application-specific minimum bandwidth. The elastic task model is highly qualified for applications that can take advantage of higher computational bandwidth to produce improved results or realize additional functionality. It extends the well-known periodic task model, which is inaccurate in case of large variations of execution times.

An example illustrated situations, in which a dynamic bandwidth assignment improved the performance of VMs compared to a static scheme by redistributing capacity. The necessary validation is still on-going and includes the integration into a real-time multi-core hypervisor and a paravirtualization of a real-time operating system.

Acknowledgment

This work was partially supported within the project ARAMiS by the German Federal Ministry for Education and Research with the funding IDs 01IS11035, by FEDER through the COMPETE program, and by the Portuguese Government through FCT grants SENODS - CMU-PT/SAI/0045/2009 and SMARTS - FCOMP-01-0124-FEDER-020536. The responsibility for the content remains with the authors.

References

[1] A. Monot et al., "Multicore Scheduling in Automotive ECUs," in *Embedded Real Time Software and Systems*, 2010.

[2] Z. Gu and Q. Zhao, "A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization," *Journal of Software Engineering and Applications*, vol. 5, no. 4, pp. 277–290, 2012.

[3] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," in *Journal of the ACM*, 1973.

[4] S. Vestal, "Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance," in *Real-Time Systems Symposium*, 2007.

[5] R. Marau et al., "Efficient Elastic Resource Management for Dynamic Embedded Systems," in *Conference on Trust, Security and Privacy in Computing and Communications*, 2011.

[6] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proc. of the 24th Real-Time Systems Symposium*, 2003.

[7] A. Mok and A. Feng, "Real-Time Virtual Resource: A Timely Abstraction for Embedded Systems," in *Lecture Notes in Computer Science*, vol. 2491, 2002, pp. 182–196.

[8] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: Response-time analysis and server design," in *Conference on Embedded Software*, 2004.

[9] G. Lipari and E. Bini, "Resource Partitioning Among Real-Time Applications," in *Euromicro Conference on Real-Time Systems*, 2003.

[10] S. Baruah et al., "Mixed-Criticality Scheduling: Improved Resource-Augmentation Results," in *Conference on Computers and Their Applications*, 2010.

[11] D. de Niz et al., "On the Scheduling of Mixed-Criticality Real-Time Task Sets," in *Real-Time Systems Symposium*, 2009.