1. GI/ITG KuVS Fachgespräch „Virtualisierung“

Paderborn Center for Parallel Computing PC²
Universität Paderbon
11.-12. Februar 2008

André Brinkmann, Holger Karl (Editoren)

Liste der Teilnehmer

Karsten Beins	Fujitsu Siemens Computers GmbH
Georg Birkenheuer	PC²
Roland Bless	Universität Karlsruhe
Stefan Boettger	Kirchhoff-Institut für Physik
André Brinkmann	Universität Paderborn
Sven Brütt	Fujitsu Siemens Computers GmbH
Christopher Dobroschke	AMD
Hubert Doemer	Universität Paderborn
Sascha Effert	Universität Paderborn
Ali Fesi	Universität Tübingen
Xiaoming Fu	Georg-August-Universität Göttingen
Christoph Geisler	
Carmelita Görk	Universität Bremen
Henning Haesken	Universität Paderborn
Jens Hagemeyer	Universität Paderborn
Bernhard Homölle	Fujitsu Siemens Computers GmbH
Matthias Hovestadt	TU Berlin
Robert Kaiser	Fachhochschule Wiesbaden
Holger Karl	Universität Paderborn
Simon Kastenmüller	Fujitsu Siemens Computers GmbH
Timo Kerstan	Universität Paderborn
Markus Köster	Imperial College, London, UK
Reihold Kroeger	Fachhochschule Wiesbaden
Marcel Kunze	Forschungszentrum Karlsruhe
Tobias Lindinger	Ludwig-Maximilians-Universität München
Dan Marinescu	Fachhochschule Wiesbaden
Mario Mense	Universität Paderborn
Gudrun Oevel	Universität Paderborn
Hans P. Reiser	Universidade de Lisboa
Gunnar Schomaker	Universität Paderborn
Bernhard Schräder	Fujitsu Siemens Computers GmbH
Jens Simon	PC²
Anke Spiegel	Hochschule Fulda
Katharina Stahl	Universität Paderborn
Martin	NEC Laboratories Europe
Kerstin Voss	PC²
Andreas Wundsam	TU Berlin
Yasir Zaki	Universität Bremen

Inhaltsverzeichnis

Hardware, CPUs und eingebettete Systeme

Netzwerk-Virtualisierung

HPC-Virtualisierung und Grids

Management

Implementierungen

Speichervirtualisierung

Hardware, CPUs und eingebettete Systeme

Applying Virtualisation to Real-Time Embedded Systems

Robert Kaiser
Distributed Systems Lab,
University of Applied Sciences, Wiesbaden, Germany
kaiser@informatik.fh-wiesbaden.de

Abstract

Virtualisation has recently received increasing attention. This new interest is caused mainly by the new applicability of the technology to desktop computers. Embedded systems, however, have so far hardly been regarded as a viable target for virtualisation, although such an approach would definitely make sense: Software for embedded devices today often comes from different vendors requiring different operating system interfaces and it has generally reached a level of complexity comparable to that of desktop applications. Embedded systems are now facing many of the same problems that once initiated the consolidation movement in the server world. Thus, it seems only logical to apply the technology that has worked so well for servers to embedded devices now. However embedded systems also have some requirements which are new to virtualisation: Most notably, there are real-time applications that must show deterministic timing behaviour.

In this contribution, we concentrate on the problem of achieving temporal determinism with virtual machines (VMs). We will give an estimation of the impact that virtualisation has on timely execution of programs, we present measurement results showing the temporal behaviour of the Xen virtual machine monitor, and we suggest some approaches how – taking into account the typical requirements of real-time programs – better timing predictability can be achieved.

1 Introduction

Virtualisation has received increasing attention during the past four years, both from academia as well as practitioners. The reason for this new interest in a technology which was invented almost forty years ago is mainly due to its new applicability to non-mainframe computers. Products like Xen [2] and VMware [10] enable multiple operating systems to coexist securely within a single machine and nowadays both are widely used in the area of server consolidation as well as on desktop machines.

Embedded systems have so far hardly been regarded as a viable target for virtualisation, although applying the technology in this field would also stand to reason: Embedded applications have reached a level of complexity equal to that of many desktop applications. This increased complexity brings new challenges to the embedded world: Safety and security requirements call for securely isolated subsystems, applications from different vendors may require different (sometimes even contradictive) operating system functionalities, yet they must coexist (and cooperate) in a single physical machine. Embedded systems are now facing many of the same challenges that once initiated the consolidation movement in the server world. Moreover, most modern embedded system hardware is capable of supporting virtualisation. Thus, it seems only logical to apply the technology that has successfully worked for servers to embedded devices now.

However, when using virtualisation in embedded systems, some new problems emerge that were not present in the field of server consolidation. Most notably, there frequently exist real-time applications that must show deterministic timing behaviour.

2 Requirements of complex embedded systems

Embedded systems have seen a steady increase in computational resources over the last years. Today, even lower-end embedded systems have processing capacities that not so long ago would have required a workstation to provide. The relation between the cost of the computing components versus the overall system cost has reached a point where cutting down on memory and processor performance no

longer yields a significant cost reduction. Yet, even lower-end systems are capable of covering a range of functionalities that would formerly have required a multitude of separate embedded control units, and, for the sake of cost effectiveness, that is exactly what they should be used for [4, 9].

Besides yielding more functionality at less cost, reducing the number of nodes in a distributed system of embedded controllers also promises better reliability, because the overall complexity of the hardware is reduced. However, the complexity of software per system increases. As a result, we are facing some new challenges:

- **Diversity of operating system interfaces:** Specific operating system interfaces are chosen because they reflect the requirements of the applications that use them. When multiple different applications are consolidated into a single system, many of them will typically bring along their own idea of what the operating system's functionality should be. This presents a problem because, traditionally, there exists only one operating system interface per machine. To define an interface that will equally fit all application's needs is difficult, and, even if such a versatile operating system can be found, all programs will need to be adapted to it. This is a costly task, especially for legacy code.
- **Need for fault containment:** When multiple functions are integrated into a single system, a fault in one of them can potentially affect all other functions. This is unacceptable in a situation where the functions are logically independent. The corresponding programs may even come from different vendors and may be completely unaware of each other. Clearly, a mechanism is needed, which ensures that any fault remains contained within the domain in which it occurred.

These problems have been faced in the field of server consolidation before and virtualisation has been a successful response. Embedded systems, however, also have to support real-time applications. This is a new requirement for virtualisation: The spatial and temporal *separation* between virtual machines does not suffice, spatial and temporal *determinism* are also required.

The system workload of an embedded system typically consists of of a mixture of applications with a very broad range of timing requirements. There will usually be:

- "hard" real-time processes for which even the slightest violation of deadline must be considered a fatal error,
- "soft" real-time processes for which it is generally desired to complete within a deadline, but exceeding it occasionally is not considered harmful,
- "non-real-time" processes that do not have timing requirements at all. Instead, these processes are expected to evenly share their computational resources, and to maximise system utilisation.

Real-time processes (whether "soft" or "hard') will generally fall into one of two sub-categories:

- Time-driven: Program execution is controlled by a static schedule. Processes are activated at predetermined points in time and run for predetermined amounts of time. In all practical cases, the schedule will be repeated periodically, i.e. these are also periodic processes.
- Event-driven: Processes are activated in response to events which occur at unpredictable points in time. Although unpredictable, the maximum rate, at which events can possibly occur, must be assumed to be bounded, otherwise a system overload could not be prevented, i.e. these are also *sporadic* processes.

Both approaches have their specific advantages and disadvantages, and combining both in a single system is generally problematic [3].

Whether real-time requirements are considered "soft" or "hard" is more a question of service quality than a conceptual one. However, whether or not processes are designed to meet real-time requirements at all does have a significant impact on the concepts they use.

3 Virtualisation impact on timing behaviour

A virtualisation platform that supports multiple VMs needs to define some method of scheduling according to which it switches between them. This *virtual machine scheduler* is typically, but not necessarily, an integral part of the virtual machine monitor (VMM) that implements the VMs. From its point of view, VMs are just processes. It is oblivious to the fact that these processes might internally run operating systems which – again – switch between processes. The goal of a virtual machine monitor is to give each VM the illusion of having the resources of a complete physical machine at its disposal, while these resources are really only subsets of a physical machine. Processing capacity is one of those resources, thus each of the VMs should receive a proportional share of the total processing capacity.

Ideally, this share would be evenly distributed over time, however, with a single processor[1], a virtual machine scheduler can only allocate time periodically to the VMs in units of finite time slices, so the actual relationship between the VM's execution time and real world time can only approximate this idealised behaviour (see the solid lines in Figure 1).

[1] or, more generally: when the number of VMs exceeds the number of processors

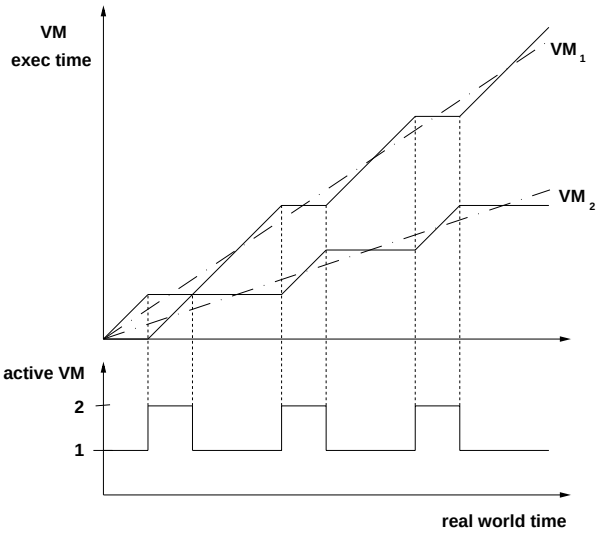


Figure 1. VM execution time vs. real time.

To estimate the impact of virtualisation on the temporal behaviour of processes that run inside a VM, we consider a simplified example: We assume a system with N VMs, all of which are periodically active for the same duration, T_{vm}. The time it takes to switch between VMs is considered to be constant, T_{sw}. The period of the VM schedule is $p = (T_{vm} + T_{sw}) \cdot N$. Every VM experiences a "blackout period", i.e. a periodically repeated interval of inactivity where none of its processes can run. The duration of this time interval is:

$$T_{del} = T_{vm} \cdot (N - 1) + T_{sw} \cdot N \tag{1}$$

Since we can not expect process activity inside the VM to be correlated with the VM scheduling cycle, we must assume that this "blackout" can hit a process at an arbitrary point in time: if, for example, it occurs between a process' arrival time and its start time, then the process' response time is increased by T_{del}, which implies that the process' jitter, i.e. the worst case distance between a process arrival time and its start time is also increased by T_{del}. If the "blackout" occurs while a process is running, then its computation time will be increased by T_{del}, so any deadline the process has to meet must be increased by the same amount. Thus, virtualisation has an impact on all process parameters which describe its real-time performance.

To reduce this impact, it would obviously be desirable to make the per VM time allocation, T_{vm} as small as possible, but there is a practical limit due to the inherent cost of switching between VMs: The context switching time T_{sw} remains constant (it is basically a hardware property). Thus, when making T_{vm} smaller, more time is wasted by context switches. The relative performance overhead caused by switching between VMs is

$$U_{vm} = \frac{N \cdot T_{sw}}{N \cdot (T_{vm} + T_{sw})} = \frac{T_{sw}}{T_{vm} + T_{sw}} \tag{2}$$

Thus, the worst case impact T_{del} expressed by context switch time and overhead is:

$$T_{del} = T_{sw} + \frac{T_{sw} \cdot (N - 1)}{U_{vm}} \tag{3}$$

(In this equation, the context switch time is a property of the computing hardware, while the acceptable switching overhead would be up to the system designer to decide.)

Comparing this to the situation of a real-time system that runs on a real machine instead of a virtual one reveals a severe drawback: Here, the achievable response time and jitter are directly limited by the context switch time, T_{sw}. Therefore, the relative impact of virtualisation on worst case response time (and also jitter) of a VM-hosted system, i.e. the ratio between the worst case response time of a virtualised program and that of a non-virtualised, but otherwise equivalent program, is:

$$\frac{T_{del}}{T_{sw}} = 1 + \frac{N - 1}{U_{vm}} \tag{4}$$

To give a realistic example: A system with three VMs and an accepted switching overhead of 5% will exhibit a worst case latency which is 41 times higher than that of an otherwise equivalent, non-virtualised system. This impact is quite severe, nevertheless, it is bounded.

4 Experiment: Xen

In order to obtain a realistic picture of the real-time functionalities that current virtualisation systems are able to provide, an experiment was made with the Xen virtual machine monitor: A Xen system hosting two virtual machines was configured. The first VM ("Dom0") hosted a typical Linux System, while the second VM ("DomU") hosted a periodic real-time process. This real-time process was programmed to cause a constant load, i.e. to actively consume – in the shown case – 60% of its time slice. The period of the real-time process was varied in steps, while the Linux system in Dom0 was kept either idle or fully loaded by an application program (busy loop). In this configuration, the latency of the real-time process was measured.

Figures 2 and 3 show the results of the experiment. The line indicated as "limit" shows the borderline above which the sum of the measured latency and the real-time process' execution time is greater than its period. For all points above this line, the deadline has been exceeded, i.e. real-time requirements can not be met.

The measured latencies appear[2] to be bounded, implying that real-time operation is possible in principle. However,

[2] This is only an experiment, not a proof

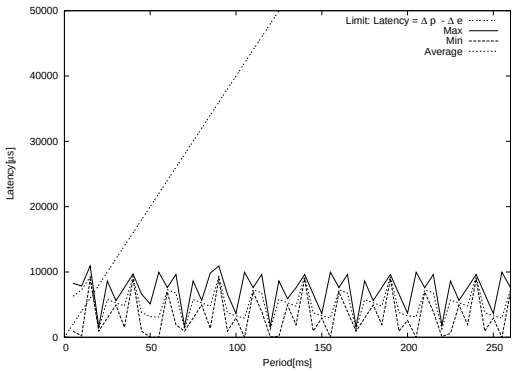


Figure 2. Xen process latency (Linux idle).

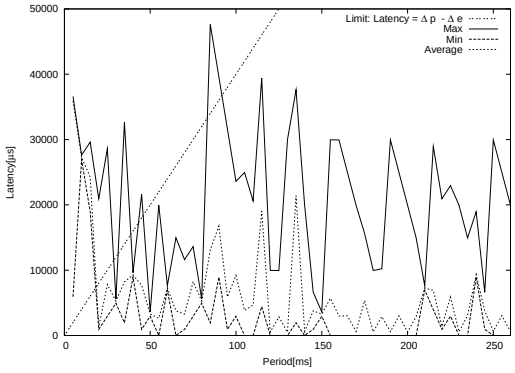


Figure 3. Xen process latency (Linux busy).

the observed latency values exceed those of contemporary, non-virtualised real-time systems by roughly three orders of magnitude.

Moreover, a comparison of figures 2 and 3 demonstrates that, even though the domains may be spatially decoupled, there is clearly a strong temporal dependency between them: wether the Linux system in Dom0 was idle or busy had a strong impact on the temporal behaviour of the real-time process in DomU.

5. Requirements for real-time virtual machine scheduling

In section 3, we estimated the impact of virtualisation on critical real-time properties of processes such as response time, jitter and worst case computation time. This impact turned out to be quite extensive, nevertheless, it is a bounded value. The experimental data shown in section 4 confirm this assertion qualitatively, although the values measured for the Xen virtual machine monitor are even worse than what our simple model would have predicted. At any rate, we can say that programs running in a virtualisation environment are in principle able to meet real-time requirements, however, their real-time performance as measured by jitter and shortness of the deadlines a system is able to meet, will always be worse than that of an equivalent program running on a physical machine by several orders of magnitude.

Whether or not this is acceptable depends on the requirements of the real-time application (e.g. the process being controlled). There are many practical use cases where response times and jitters in the range of tens or hundreds of milliseconds are perfectly adequate. Such applications can thus be straightforwardly executed under a virtualisation environment. However, especially in the fields of automotive, avionics or industrial control, there exist many applications for which such latencies are simply unacceptable.

If virtualisation is to succeed as a generic approach to embedded system consolidation, the requirements of such high-performance real-time applications have do be considered. Summarising the requirements outlined in section 2, a virtualisation system should be prepared to handle three conceptually different classes of processes: time-driven and event-driven real-time processes as well as non-real-time processes. The functionalities that these process classes require their operating system to provide are largely (though not entirely) orthogonal. Although it is conceivably possible for an operating system to support typical real-time as well as non-real-time functionalities with a single programming interface, such a system will in practice tend to be either unnecessarily complex or to be insufficient from either the real-time or the non-real-time perspective. Therefore, in a virtualisation environment, whose main ability is to support multiple operating systems, each class of process should be able to use its own, dedicated operating system interface. Therefore, our approach assumes that different process classes will each exist in separate VMs: We assume that a virtual machine environment for embedded systems will have to schedule VMs which are explicitly time-driven, event-driven, or non-real-time VMs as a whole. For each VM class, different requirements are to be fulfilled by the VM scheduler:

1. **Determinism:** For time-driven processes to be able to execute at their predetermined points in time their VMs must be active whenever any of them is active, i.e. the VM schedule must be matched against the schedules of the time-driven processes. More formally, with $\sigma_i(t)$ being the schedule for the processes of the i-th VM and with $\sigma_i(t) \neq 0$ for any point in time t when any of its processes is running, the VM schedule for all time-driven VMs is:

$$\sigma_{vm}(t) = \begin{cases} i, & \forall t \in R^+, i \in N^+, \sigma_i(t) \neq 0 \\ 0, & \forall t \in R^+, i \in N^+, \sigma_i(t) = 0 \end{cases} \quad (5)$$

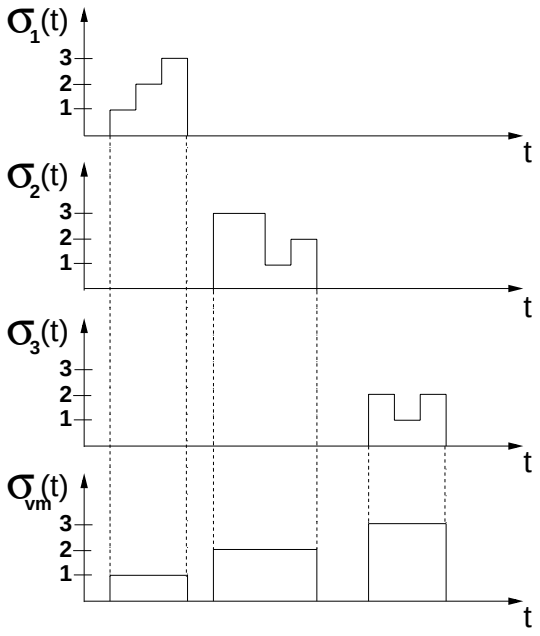


Figure 4. "Enclosing" VM schedule for time-driven virtual machines.

i.e. the VM schedule "encloses" the time-driven schedules of the VMs (see Figure 4). This assumes that the time-driven schedules do not overlap. Also, if they are periodic (as they are in most practical cases), they must all use the same period. Since all time-driven process schedules are predefined, the same also applies to the enclosing VM schedule: The schedule for the time-driven class of VMs must be strictly a function of time only.

2. **Responsiveness:** For event-driven real-time processes, the prime requirement is their ability to react on an external event within a deterministic (and preferably short) amount of time. However, if the process handling such an event runs within a VM, its reaction can only take place while that VM is active. With VMs being activated cyclically according to a predefined schedule as the previous requirement demands, the considerations of section 3 apply here, i.e. worst case response time, worst case computation time and jitter are predictable, but they may simply be too long in some situations. These potential problems motivate a second requirement, namely that there should be a way at least for select, privileged VMs to respond to events outside of the time-driven schedule, i.e. to preempt the currently running VM.

3. **Dynamic re-allocation of excess computing time:** In real-time scheduling, it is generally necessary to allocate time to processes according to worst-case assumptions. However, worst-case scenarios only apply rarely and in the average case, real-time processes often have more than sufficient time to complete their jobs. As a result, real-time systems tend to exhibit a low average processor utilisation. This is considered to be the "price of real-time". But if there also exist non-real-time processes in the same system, these excess computational resources could be put to good use by passing them on to those non-real-time processes. Since we assumed above that real-time and non-real-time processes exist in different VMs, the virtual machine scheduler would be the place to implement this dynamic time re-allocation policy.

6 Approach

The first of our requirements (maintaining a deterministic VM schedule) is straightforwardly accomplished by using a strictly time-driven scheduler: Every VM is statically assigned an individual time slice. The virtual machine scheduler periodically executes each VM in turn for the duration of their respective time slices.

In this way, VMs receive fixed amounts of processing capacity at predefined points in time. Thus, they are able to schedule time-driven real-time processes themselves. However, if a VM has no runnable processes during its active time slice, or if its processes have completed before the time slice is over, it can not simply do a switch to another VM, because that would destroy the temporal determinism. All it can do is "burn away" the unused time. Figure 5 shows an example: The VM schedule $\sigma_{vm}(t)$ assigns time slots to three VMs. These time slots are dimensioned according to the VM's respective worst case computation times, T_{wce1}, T_{wce2} and T_{wce3}. Since these are worst case times, the actual execution will usually finish earlier. In Figure 5, $\alpha_{vm}(t)$ shows an example of the actual VM activities. The accumulated unused computation time is shown in Figure 5 as $e_{exc}(t)$.

To enable re-use of these excess processing capacities, we combine time-driven scheduling with priority-based scheduling: In addition to a time slice duration, we also assign a priority to each VM. To all the time-driven real-time VMs, we assign (possibly different) mid- or high-level priorities. We continue to switch between these VMs according to a strictly time-dependent schedule. All non-real-time VMs are assigned the same low-level priority. Switching between these VMs is done by a classical round-robin scheduler to achieve load balancing. Thus, whenever the currently active time-driven real-time VM has no processes to run, it can sleep for the rest of its time slice, effectively passing its excess processing time to all low priority (presumably non-real-time) VMs, which will share it evenly.

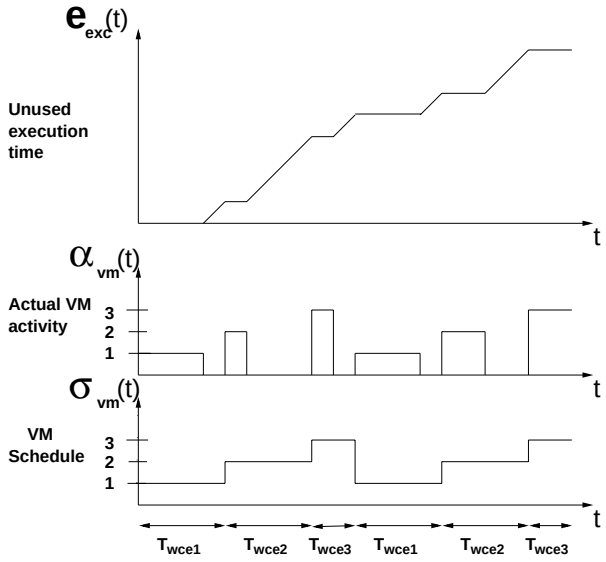


Figure 5. Strictly time-driven VM scheduling and unused processing time.

Figure 6 shows an example: The low-priority VM schedule ($\lambda_{vm}(t)$) is activated whenever time is not allocated to or used by the mid- and high-priority VMs. This accomplishes our third requirement.

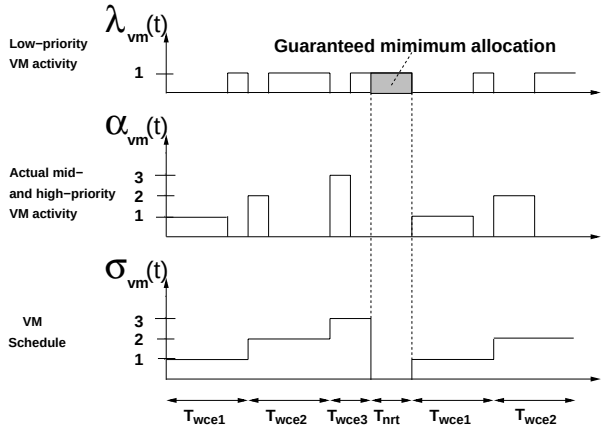


Figure 6. Combined time and priority-driven scheduling.

In order to prevent starvation of the non-real-time VMs in cases where the real-time VMs actually do consume all of their allocated time, we can assign a certain minimum time allocation to the non-real-time VMs, simply by leaving a portion of the time-driven scheduler's cycle unallocated. Figure 6 also shows this: In this example, the non-real-time VMs will always have at least a time of T_{nrt} at their disposal.

With the features introduced so far, an event-driven real-time VM that needs to activate processes in response to asynchronous events can only do so during its allocated time slice. As explained earlier, this leads to rather long worst case response times and jitter. The effects can be reduced to some extent by invoking the VM at a higher rate, either by increasing the cycle frequency, or by allocating multiple time slices per cycle to the response-critical VMs. However, the resulting frequent context switches cause increased overhead.

If the requirements regarding response time and/or jitter can not be met using these methods, the one remaining possibility is to assign a sufficiently high priority to the VM in question, thus enabling it to preempt other, lower priority VMs at any time. Such a feature must be used with care, though, because high priority VMs effectively "steal" their processing time from whichever other VM happens to be active at the time the triggering event occurs. In a worst-case scenario, all high-priority VMs that exist in the system could be triggered at the same time, so the worst case amount of "stolen" time would be equal to the combined worst-case computation times of all high-priority VMs. This potential amount of time that may possibly be consumed by high-priority VMs must be known in advance, and it must be bounded. Therefore, only a select group of well-trusted VMs whose worst case computation times are known can be granted the privilege of a high priority. To compensate for the potential loss of time, all lower priority real-time VM's time allocations must be increased by this worst-case time amount. If this rule is obeyed, the mid-priority VMs will be able to meet their deadlines even in the presence of high-priority event-driven VMs. However, they will also exhibit increased jitter. Despite its slightly unattractive properties, we feel that the concept of high-priority VMs is generally needed to enable fast, deterministic reaction to external events (i.e. interrupts). It satisfies our second requirement.

The problem is actually a classical example of the dichotomy between the time-driven and the event-driven approach: whenever both are combined in a system, one of them has to be given precedence and as a consequence, the other is destined to perform poorly. This cannot be solved in a single processor environment. Nevertheless, our approach is flexible enough to allow the choice of precedence to be made individually for every time-driven virtual machine: Since every VM (whether time-driven or event-driven) can be assigned an individual priority level, every time-driven VM is able to select the event-driven VMs which are able to preempt it, simply by choosing an appropriate priority for itself with respect to the priorities of the event-driven VMs in the system.

Whether or not use of this functionality is necessary de-

pends on the potential delay in response time that a given application can tolerate and also on the switch rate that the system can achieve while maintaining an acceptable overhead. This overhead is partly a feature of the underlying computer architecture[3], but it also depends on the effectiveness of the scheduling mechanism. In the next section, we describe a practical implementation of the methods outlined so far.

7. Implementation

The scheduling method outlined in the previous section is in practical use today as part of "PikeOS", a commercial product [5]. PikeOS is a microkernel-based runtime environment targeted at embedded systems. It is able to host multiple real-time and non-real-time guest operating systems, guaranteeing to each of them their own set of temporal as well as spatial resources. The system uses only a very small amount of trusted code (mainly the microkernel) to implement this separation policy.

The PikeOS microkernel is conceptually based on the ideas introduced by Jochen Liedtke's microkernel L4 [6, 7]: Activities are represented as threads and each thread is linked to a "task" which serves as a container for rights to access different kinds of resources. All threads that belong to a given task have access to these resources. Similar to L4, threads are assigned a static priority level, but unlike L4, they are also grouped into sets which we refer to as "time domains", τ_i. The microkernel supports a configurable number of such time domains. Each of them is represented in the microkernel as an array of linked lists (called *ready queues*), with one list per priority level (see Figure 7). Threads that have the same priority level are linked into the same list in a first in/first out manner. The thread at the head of the list is executed first. When a thread blocks, it is appended to the end of the list. So, within a time domain, there is a typical, priority-driven scheduling with round robin scheduling between threads at the same priority level, as it is used in many real-time operating systems today. However, unlike those systems, the PikeOS microkernel supports multiple time domains instead of just one. Threads can only execute while their corresponding time domain is active, regardless of their priority. If we cycled through the time domains, activating each one at a time for a fixed duration, we would obtain the exact behaviour of an ARINC 653 scheduler as described in [1]. But in contrast to this, the PikeOS kernel allows *two* of the time domains to be active at the same time:

- The first time domain, τ_0 plays a special role in that it is always active.
- Of all other domains $\tau_i (i \neq 0)$, only one can be active at a time. The microkernel provides a (privileged) system call to select the currently active time domain. Switching happens cyclically, according to a preconfigured, static schedule.

The microkernel scheduler always selects for execution the thread with the highest priority from the set union of τ_0 and τ_i. Figure 7 shows the principle.

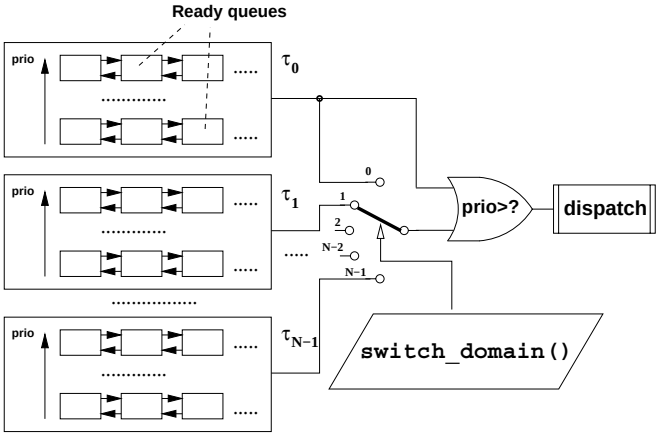


Figure 7. Principle of implementation.

In this picture, `switch_domain()` is a microkernel function that selects one of the domains $\tau_i (i \neq 0)$ to be the currently active one. The threads have different semantics, depending on their priorities and on their time domain:

- $\tau_i (i \neq 0)$: The totality of all threads in the currently active "foreground" time domain cyclically receives a fixed amount of time. Generally, these threads will be configured to have a mid- through high-level priorities (though this is technically not a requirement).
- τ_0: The semantics of the threads assigned to τ_0, the "background" time domain depend on their respective priorities. Usually, these will be either below or above those of the threads in all the possible forground time domains $\tau_i (i \neq 0)$.
  - Low-priority threads within τ_0 will receive the processing time that was assigned to, but not used by the mid-priority threads in τ_i. All non-real-time VMs are therefore implemented by threads running at the same, low priority in τ_0. Since their priorities are equal, they run under a round robin scheduler, sharing their amount of computation time evenly.
  - High-priority threads in τ_0 can preempt any low- or mid-priority threads at any time. This is used to implement the event-driven, high-priority VMs described in the previous section.

[3] Depending on cache and TLB structure, the cost of a context switch can vary significantly between different architectures.

The microkernel implements only the mechanism[4] to select one of the time domains as active foreground domain. The policy part of deciding which of the domains is activated when is left to the user level. This can be done by an interrupt handler thread which runs at a high priority in τ_0. In a typical configuration, this thread gets activated by an external one-shot timer whenever the timeslice of the currently active foreground domain has expired. It then activates the next time domain, re-programs the one-shot timer to trigger an interrupt when the next domain's time allocation expires and then waits for the interrupt. However, this is only one possible example: Since it is implemented at the user level, the domain switching policy can easily be replaced without any changes to the microkernel.

Some initial experiments with a PowerPC platform[5] have shown worst case context switch times to be in the range of 25μs. Therefore, if an application can live –for example– with a context switch overhead of 10%, minimum per domain time allocations can be made as low as 225μs.

8. Conclusion and outlook

In this paper, we estimated the impact of virtualisation on the real-time properties of programs and we analysed the requirements for using virtual machine schedulers in embedded systems. Looking at the Xen virtual machine monitor as an example, we showed that it does not fulfil these requirements and is thus of limited use in its current form. We outlined a scheduling method for switching between VMs with varying degrees of real-time requirements. This method enables hard real-time and non-real-time virtual machines to coexist both safely and effectively in a single machine.

Current research aims at extending the scheduling method for use in both multiprocessor as well as distributed systems. This opens up a number of interesting new prospects:

- Separation of time-driven and event-driven real-time programs: The presented method can alleviate some of the problems that tend to emerge whenever time-driven and event-driven programs are mixed. Yet, as we have seen, it can not completely eliminate those problems. They could be solved, however, in a multiprocessor system by binding the time-driven and event-driven threads to separate processor cores.

- Coscheduling of parallel applications: Parallel programs execute as multiple, tightly interacting threads. In order to live up to their potential, these threads need to execute at the same time on different processor cores [8]. The system should support this by identifying threads which are part of a parallel program and by ensuring that they are always executed simultaneously.

- Coscheduling of distributed applications: In a distributed system, there is often a need for multiple programs residing on different nodes to act synchronously. From the scheduler's point of view, this is similar to the coscheduling of a parallel program's threads, except that here the threads exist on different nodes. Therefore, the schedulers on all the nodes in the distributed system need to have a common notion of time. Furthermore, the communication channels used by the threads to interact with each other need to be taken into consideration when making scheduling decisions. So, in addition to temporal and spatial resources, the system will also have to provide guaranteed communicational resources (e.g. network bandwidth) to its VMs.

[4] Called `switch_domain()` in figure 7

[5] Motorola MPC5200 at 400 MHz.

References

[1] ARINC. Avionics Application Software Standard Interface. Technical Report ARINC Specification 653, Aeronautical Radio, Inc., 1997.

[2] P. Barham, B. Dragovic, K. Fraser, S. H, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization, 2003.

[3] G. Fohler. Flexible Reliable Timing - Real-Time vs. Reliability. In *Keynote Address, 10th European Workshop on Dependable Computing*, 1999.

[4] R. Greene and G. Lownes. Embedded CPU target migration, doing more with less. In *TRI-Ada '94: Proceedings of the conference on TRI-Ada '94*, pages 429–436, New York, NY, USA, 1994. ACM Press.

[5] R. Kaiser and S. Wagner. Evolution of the PikeOS Microkernel. In I. Kuz and S. M. Petters, editors, *Proceedings of the first international workshop on mikrokernels for embedded systems MIKES 2007*, pages 50–58, Sydney NSW 2052, Australia, January 2007. MIKES, National ICT Australia.

[6] J. Liedtke. On μ-Kernel Construction. In *SOSP*, pages 237–250, 1995.

[7] J. Liedtke. L4 reference manual - 486, pentium, pentium pro, 1996.

[8] J. K. Ousterhout. *Partitioning and Cooperation in a Distributed Multiprocessor Operating System: Medusa*. PhD thesis, Computer Science Department, Carnegie-Mellon University, apr 1980.

[9] D. Stepner, N. Rajan, and D. Hui. Embedded application design using a real-time OS. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 151–156, New York, NY, USA, 1999. ACM Press.

[10] VMware. VMware ESX Server Online Documentation, 2005.

Multi Root I/O Virtualization (MR-IOV)

Bernhard Homölle, Bernhard Schräder, Sven Brütt
Fujitsu Siemens Computers GmbH
{Bernhard.Homoelle,Bernhard.Schraeder,Sven.Bruett}@fujitsu-siemens.com

Abstract

*This article introduces architectural-wise the Multi Root I/O Virtualization (**MR-IOV**) approach as being released by the PCI-SIG® group. It also prospects about technical advantages, weaknesses and respective implications coming along with technology adoption especially to blade server systems.*

1. Introduction

Server Virtualization these days gets high awareness in the industry, as the technology brings significant benefits in respect to hardware resource utilization, security of operation, re-locatability and availability of complete OS installations.

HW technologies in chipsets and CPU's have been introduced in the last years, to simplify and to improve efficiency of Virtual Machine Management Software. Significant hardware technology milestones in the evolutional process of improving performance and security of virtual operation have been introduced:

- Instruction Set architectural extensions for virtual operation to the processors
- Memory Management Unit (MMU) Extensions supporting concurrent multiple virtual System Image **(SI)** operation

Improved hardware resource utilization, better power utilization, possibilities to migrate system instances to different hardware platforms are main features driving the demand for server virtualization. Still a fundamental pain to widely deploy Virtual Machines in datacenters is I/O performance.

In virtualized environments, multiple Operating System Instances (SI's) may share single physical I/O endpoint devices. Virtual Machine Management Software has to handle and to switch all data and message transfers between different Guest OS's and their assigned I/O devices. This is very time consuming, especially as the Virtual Machine Management Software has to ensure communication stream protection among separate virtual System Images for data and message transfers.

In early 2007 the *PCI-SIG® group* has introduced a ***S**ingle **R**oot **IO V**irtualization Specification **(SR-IOV)*** with the intention to address the respective I/O performance and security issues. A set of architectural elements has been specified to lower latency, to lower effort and to improve security for I/O data and message transfers in virtualized system environments. Major functional elements of the *SR-IOV* specification to improve I/O performance in virtualized environments:

- Address Translation and Protection Table (*ATPT*). An IOMMU which ensures that traffic initiated by a certain I/O device can access memory regions of the associated System Image, only.

- Multiple *Virtual Functions (VFs)* associated with an I/O device *Physical Function (PFs),* to offer improved sharing capabilities for an I/O endpoint device

The implementation of ***SR-IOV*** architectural elements does enable a platform for direct communication between guest operating systems and their assigned I/O device functions. The Virtual Machine Management ***(VMM)*** software does not need to handle all kind of I/O traffic between guest OS and I/O device any more.

Beyond SR-IOV specification, the PCI-SIG® is now going to introduce architectural enhancements which intend to offer further possibilities to consolidate I/O infrastructures. This technology might fit well to Blade server systems, which host multiple root complexes within a single chassis enclosure. An initial revision of a*"**M**ulti **R**oot **I/O V**irtualization Specification"* ***(MR-IOV)*** is under work by the *PCI-*

SIG® group. The **MR-IOV** specification defines mechanisms which will allow sharing of physical PCIe endpoint devices among multiple **S**ystem **I**mages (**SI**'s), running on multiple physical Server Systems.

2. Present Techniques of I/O Virtualization in Server Systems

2.1 SW based Virtualization of Server I/O Infrastructure

In traditional server configuration, PCIe endpoint

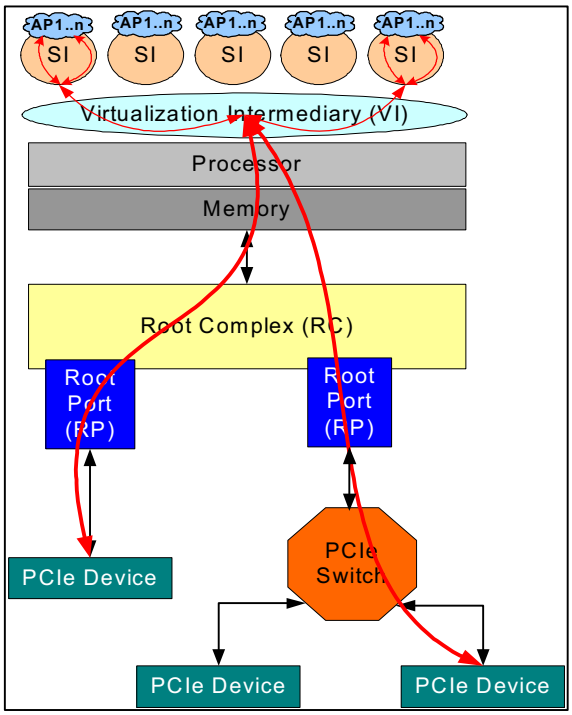


Figure 1: Virtualized Server Infrastructure

devices are under control of the Virtualization Intermediary (VI). All I/O that is coming from the various System Images (SI), is switched and managed within the Virtual Intermediary (VI), which takes full responsibility for all I/O related actions.

The VI has to ensure that application specific I/O data and message transfer needs to be isolated between different applications for security reasons.

Compared to native Server System installations, virtualized servers show basic I/O performance issues as there is need to handle several tasks in the VI by software. E.g. for storage traffic a VI needs to copy and to protect outgoing block transfers from a Guest SI into the VI. The VI needs to switch storage data transfers among multiple SI's then. For incoming traffic the VI has to figure out about incoming destination addresses and to copy blocks of storage data into the right destination guest SI then.

To consider latency impact we did look at response times for block I/O read requests in different kind of virtualized environments. The lowest storage block transfer latency impact due to virtualization has been observed running a scenario with para-virtualized drivers on guest OS'es.

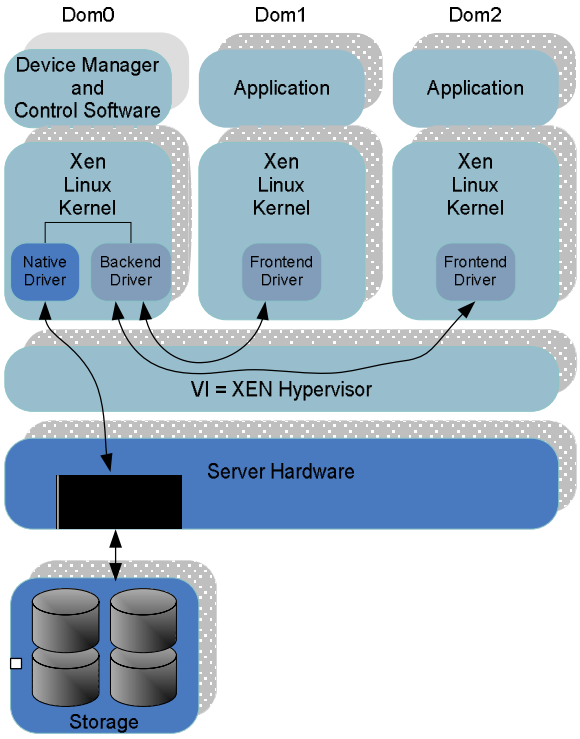


Figure 2: Virtualized Storage Data Access

The chart below compares average response times for a 512 byte block read (non-random) from storage in virtualized scenario against response times in a native, non virtualized configuration.

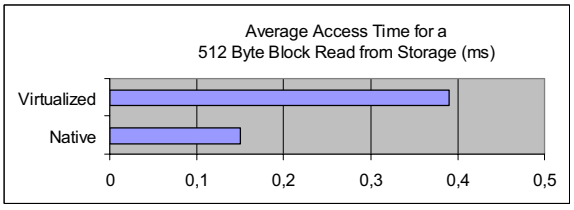


Figure 3: Block Read Access Times

For larger packet sizes the relative latency difference between native and virtualized access will be lower. However, as for LAN access lots of small packets are to be transferred, the latency impact of virtualization limits network performance significantly. To address above like shown I/O access latency issues the PCI-SIG® has introduced multiple functional elements in a *Single Root IOV Specification.*

2.2 Single Root I/O Virtualization (SR-IOV)

Single-Root I/O Virtualization, as being specified by the PCI-SIG®, provides mechanisms to share physical PCIe endpoint devices among multiple SI's. The Advantage of SR-IOV over software based I/O virtualization is that there is no more need to manage all I/O traffic between SI and PCIe device through the virtual intermediary.

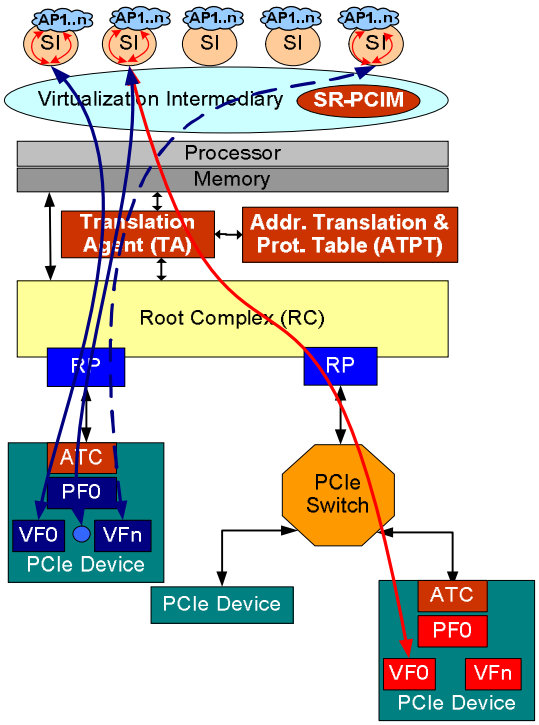


Figure 4: SR-IOV Functional Extensions

SR-IOV capable PCIe devices will offer within a single physical device a single physical function (PF) and multiple virtual functions (VF's). A VF within a PCIe device can be assigned to a related System Image. I/O data transfer between SI and related VF in a PCIe device can be handled directly without any VI interference.

Allocation of a VF in a PCIe device is done by a Single-Root-PCI Manager (SR-PCIM), which is part of the VI. A hardware based address translation and protection infrastructure as specified in the SR-IOV specification will ensure that VF initiated memory access may reach memory areas of the related SI, only.

Due to performance and security aspects being addressed by the ***SR-IOV*** specification, it looks that all major vendors of CPU, Chipset and I/O-Device technology will provide sooner or later ***SR-IOV*** technology within their products. Different parts of that technology are going to be introduced under company specific trademarks, names and specifications which make it difficult to keep an overview. The address translation and access protection mechanisms e.g. are going to be introduced by Intel as "Virtualization Technology for Directed I/O, ***VT-d"***. AMD has introduced functional similar parts under the name "Rapid Virtualization Indexing ***RVT***". RVI is part of AMD's overall Virtualization Technology being entitled "***AMD-V"***. Technical details of AMD's implementation have been disclosed as "***IOMMU*** Architectural Specification".

3. Technical Introduction to Multi Root I/O Virtualization (MR-IOV)

Beyond ***SR-IOV*** the PCI-SIG® is going to release a Multi Root IOV ***MR-IOV*** specification which enables I/O infrastructure consolidation in server system enclosures containing multiple PCI root complexes. ***MR-IOV*** supports sharing of PCIe endpoint devices among multiple physical servers. MR-IOV as being specified by the PCI-SIG®, will allow omitting PCIe devices from a physical server motherboard to save costs, power and space. Instead a software-transparent PCIe interface will be exposed from the physical compute node to access an external PCIe fabric, which provides access to sharable PCIe endpoint devices. In this approach, the virtual hierarchy of each host extends through the fabric to each PCIe endpoint device. A ***MRA-IOV*** (*Multi Root Aware IOV*) switch contains a virtual fan-out switch routed to each host port. Transaction layer packets are routed through the RC to PCI endpoint devices without any changes, which allows end to end data transport integrity check. The multi root fabric is *software transparent* only after a management and configuration agent, called MR-PCIM (*Multi Root PCI Manager*), has configured the fabric and coordinated sharing of I/O devices by dealing with resource assignment and allocation. Once this is done, each host may enumerate its own virtual

hierarchy and configure the virtual I/O devices allocated to it by MR PCIM using standard PCIe enumeration software. Device drivers may need minor tweaks but no architectural changes to support MR-IOV.

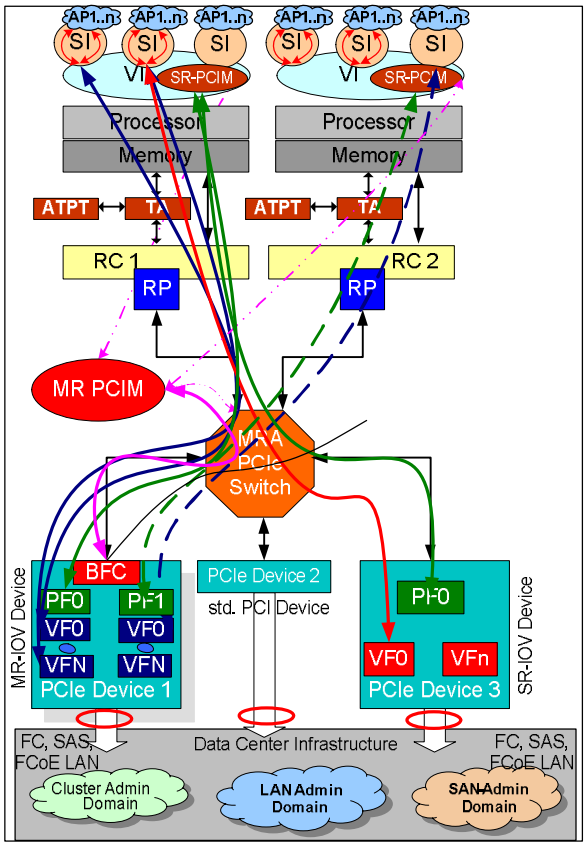


Figure 5: MR-IOV Infrastructure Overview

4. Prospects, Weaknesses and Implications upcoming with MR-IOV

The discussed aspects in this chapter may give an impression about possible implications of an MR-IOV technology adoption to blade server systems. Investigations just have started. Final conclusions can't be given, yet.

The advantage of ***MR-IOV*** is that it will allow omitting I/O-devices from a server module, particularly those for storage and networking. Instead PCIe interfaces coming out of CPU and Chipset are routed directly to a software-transparent PCIe switch fabric which forwards the I/O traffic to the PCIe endpoint devices. Below in this article we consider an adoption of ***MR-IOV*** technology to Blade Server Systems and will compare features then against the conventional I/O infrastructure of Blade Systems. We look especially at Blade Server Systems, as the ***MR-IOV*** approach seems to match well with Blade Server System architectures, enclosing *Multiple* physical *Roots* at high density packaging.

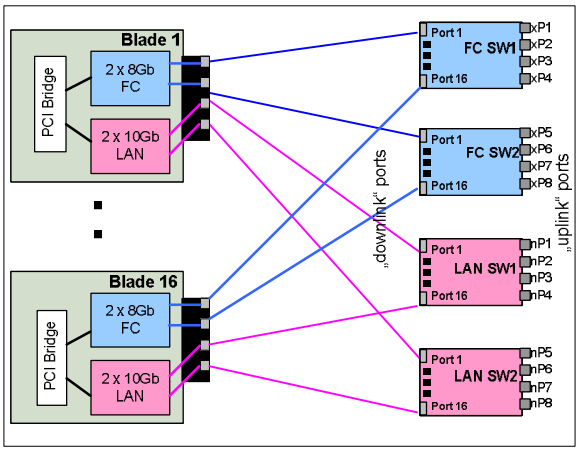


Figure 6: Conventional Blade Infrastructure

Above example of Conventional Blade Infrastructure implementation shows 16 Blades, each having a dual channel 10Gb NIC and a dual channel FC Adapter populated and being connected to redundant FC and LAN Switching Modules. By having a PCIe fabric in the MR-IOV approach, like shown in the picture below, there will be no more need to have separate types of fabrics for storage and LAN within a single blade chassis enclosure. Two PCI Fabrics are still employed in below example to ensure fabric redundancy.

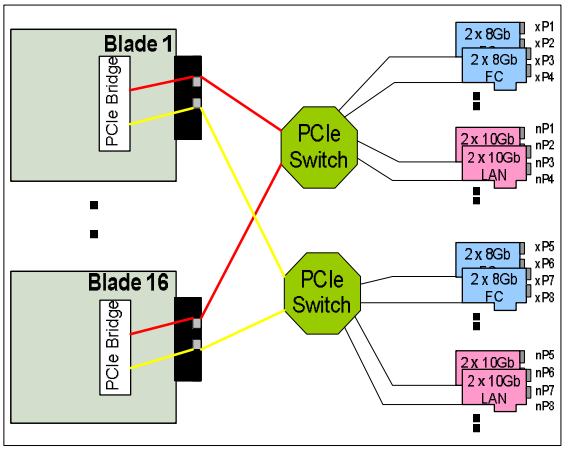


Figure 7: MR-IOV Blade Infrastructure Example

4.1 HW Cost Advantages by MR-IOV

The native capability of MR-IOV to share I/O Devices will reduce Blade Box Entry Costs. Even for typical configurations it looks that the End User Price for hardware will be lowered by making use of the MR-IOV approach.

PCIe Cards: Typically the ratio of downlink ports to uplink ports for switches is in the range of 3 to 5. As MR-IOV is making use of fractions (VFs) of adapters and doesn't require FC and LAN adapters per each blade, the overall PCIe adapter costs are expected to be a third to a fifth compared to the conventional approach, only. Looking at a time frame from 2009 to 2011, the costs for FC and 10Gb Ethernet PCIe adapters are assumed to remain a significant portion of the blade total system costs.

Switches: For a blade system which offers MR-IOV infrastructure, there will be no more need to adopt different kind of switch modules for different interface technologies like 1Gb LAN, 10Gb LAN and 4/8Gb FC to the blade housing. There will be also no more need to adopt specific switches to blade boxes, which offer e.g. unique LAN manageability features. As PCIe remains to be switched only within a blade box, the MR-IOV approach will give more flexibility in configuring type and throughput of I/O devices, while requiring a less number of switch module types. This will simplify I/O infrastructure and reduce generic blade box costs. To offer variable I/O traffic bandwidth per blade, PCIe switch units for different port width (PCIe x2, x4, x8, x16) operation might be required.

If we try by now an early estimation about HW costs for MR-IOV like PCIe HW switching in comparison to HW costs for a conventional LAN and FC switching infrastructure in a blade box, we expect to see cost advantages for the MR-IOV approach, assuming alike performing solutions. Of course there is need to dig into this more deeply based on more specific plans of implementation.

4.2 Performance

If we look at below shown elements affecting throughput and latency of I/O traffic within a blade box, the HW internal implementation of SWITCH devices will be of major relevance, comparing a "Conventional" against the MR-IOV approach.

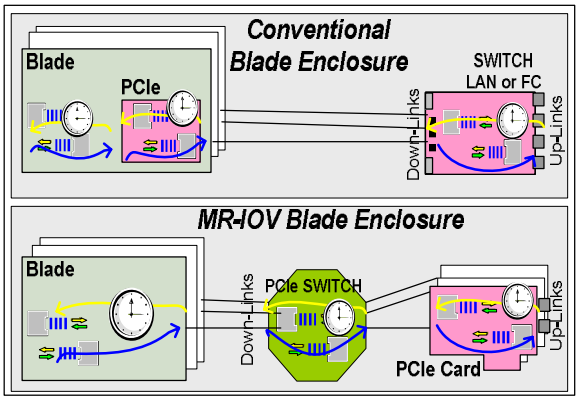


Figure 8: Blade Box IO Traffic Transfer Latencies

Latencies: The implementation of Physical Layer packet routing, congestion and buffer resource management within a switching chip device will finally make the difference in transport latency over the switch device.

For low workload conditions we wouldn't expect to see a significant packet transfer time difference, comparing an MR-IOV switch device against a 10Gb Layer2/3 LAN or even against a FC switch. To rate about packet transport delay at heavy workload scenarios, it will be essential to look at the very details of a switching device implementation.

Throughput: The MR-IOV approach allows assigning multiple I/O endpoint devices to a certain blade. This makes it possible to provide high I/O device bandwidth to individual blades. However, the maximum I/O bandwidth per blade will be limited by the number of wires and by the bandwidth per wire connecting blades to the I/O switch fabrics.

The actually released PCIe standard Revision 2.0, specifies per signal pair a maximum data rate of 5Gb/s, which is half of the speed, at which 10Gb LAN traffic is transferred per PCB routed copper wire pair (10GBASE-KR) in Blade systems today. Ethernet physical layer transfer speed per wire will continue being ahead against the per wire data rate of PCIe over the next few years even if PCIe physical layer speed will catch up in the next two years.

All this means that in the MR-IOV approach, roughly speaking a 1.5 times number of wires between switches and blades will be required to offer alike bandwidth against the conventional approach.

4.3 Power

As the I/O resource sharing capability of the MR-IOV approach typically reduces the required number of PCIe endpoint devices to a third till a fifth, the power saving will be significantly. Assuming a power consumption of 15 Watt per typical LAN or FC PCIe Card, an adoption of the MR-IOV approach could easy save 300 Watt for a fully stuffed Blade Box.

4.4 Not yet covered by the MR-IOV Spec.

Host to Host Traffic: The initial available MR IOV specification does not specify a mechanism for host-to-host communications. It looks that this complex and controversial feature was omitted to get the initial revision of the specification closed.

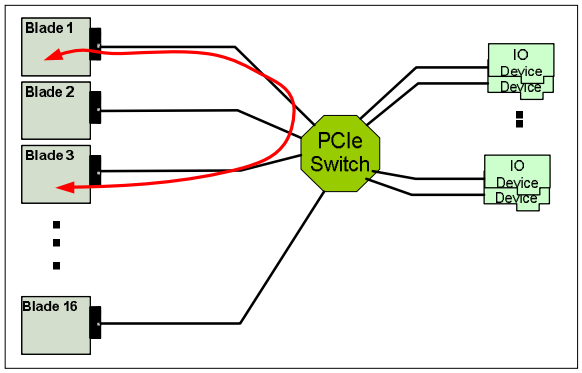


Figure 9: Host to Host Communication

For "High Availability" and "Load Balanced" Server System Configurations as well as in the High Performance Computing Market (HPC), clustering of servers has become usual. Various implementations for clustering of servers using different hardware interconnect interfaces as well as using different kind of software infrastructure models do exist.

Especially for blade systems it is a pity that the industry could not agree on a common standard to enable server clustering over cheap, low latency PCIe connections at high bandwidth. Of course there are traditional ways to cluster Blades e.g. by going through NIC devices. But this is less performing and more expensive in various matters is it could be by having respective specification in the MR-IOV spec.

4.5 MR-IOV Specification Standard

There are various aspects which need to be considered if it comes to the implementation of MR-IOV in the server environment. There is to mention the ongoing certification process for MR-IOV. Until the final specification isn't available there is some uncertainty about implementation details. However with the release of 0.9 almost all major topics are addressed. First prototypes can be built on that specification. We expect the final release of MR-IOV by the PCI-SIG® in Q1/08.

By making extensive use of 1st prototypes we have to learn more about the implementation in real server scenarios. In this sense the MR-IOV specification is just the framework for shared I/O realization. The specification itself allows different types of implementation at the Server, PCIe Controller and Management side. We have to research dependencies and opportunities of the shared MR-IOV approach along with the virtualization aspects. This includes topics like I/O stream prioritization and congestion management in the PCIe environment which will be shared between different virtual identities and even different physical servers.

- What type of interference in real server scenarios may or will happen?
- Will the VMM community support special features like VF migration or hot plugging?
- What type of interaction is required between the SR-PCIM and MR-PCIM?
- How to automate I/O assignment on application demand in virtualized and in non virtualized environments?

All these aspects are mentioned but not restricted by the MR-IOV specification and will allow vendor specific implementations. This can be seen as threat as well as a chance to get an MR-IOV ECO system established.

5 Added Values for Customers

The MR-IOV approach provides some new thoughts of how servers can be efficiently connected to the outer world including I/O management while keeping the current PCIe Controller environment. The next chapters will show some of these ideas with respect to added value for end customers.

5.1 Motion of Server Environments

New virtualization features like Xen-Motion or VMotion allow movement of virtualized environments beyond server borders. To make that efficient it is useful to prepare and manage I/O identities like MAC or WWN as well as connection parameters. Both the I/O identity and the connection parameters like necessary interfaces and throughput requirements can be efficiently assigned in combination with the MR-PCIM. It will be possible to discover conceivable destination I/O capabilities before movement takes place. This is possible without interference to running environments. Even a running server isn't required to get the I/O capabilities of a server connected to a MR switch. All this can make movements of virtualized System Images (SI) in the data center more reliable and flexible with respect to the related I/O.

5.2 Flexible Assignment of I/O

In today's server environment the I/O is fixed assigned to each server due to the physical placement of PCIe Controllers in the corresponding server I/O slots. The amount of available I/O slots and the type (FC,LAN,SAS etc.) of IO also defines possible use-cases for that server. Because I/O-Slots are expensive in terms of necessary space, power and money, the amount of available slots is therefore limited. As a consequence the customers need to decide very early in his decision process for what purpose the dedicated server will be used. Changes over lifetime are difficult and cause significant service effort. There is of course still some flexibility coming along with available PCIe slots but especially for small servers with one or two PCIe slots it is rather stiff than flexible.
With the MR-IOV model it will be simple for the customer to add or remove I/O devices (FC, LAN, SAS etc.) to each connected server based on what the application requires. The server I/O capability itself isn't any longer limited by the local available I/O slots and local plugged I/O controller. Of course the principal I/O potential of a server defined by the chipset and the amount of CPU's memory etc. remains.

5.3 Flexible and Remote Administration

Unlike today's I/O administration the MR-IOV approach allows flexible reaction on I/O problems like bottlenecks, broken connections etc. Each I/O connection to each server can be administrated without the need to handle physical PCI-cards in the data center. The administrator will simply be able to manage I/O connections from his desk. If the application runs in throughput limitations or another type of I/O will be needed, an administrator may assign further interface devices just by doing a few mouse clicks.

5.4 Redundancy and Services Concepts

Redundancy in the current rack server environment is quite expensive. You have to double the amount of necessary I/O controllers for each I/O type. That might be impossible due to PCIe-Slots limitations. The MR-IOV approach will allow redundancy via a shared PCIe connection. Furthermore the MR-IOV redundancy concept requires one VF as spare VF on a second controller only. This will save power and PCIe controllers.

Due to the MR-IOV redundancy concept, controller function faults will not require immediate service. It will be acceptable to delay service to next business day because the application can run with the redundant interface. Even if the complete throughput is required it can be smoothly redirected to another, e.g. spare PCIe controller to allow running the application with full speed.

.

5.5 Controller Savings and I/O consolidation.

One prerequisite for effective MR-IOV usage model is the over provisioning of I/O throughput coming along with the newest I/O controllers like 10GbE or future 40/100GbE and 4/8 or future 16 Gb FC. This over provisioning inspires to think about I/O controller sharing. In today's blade environment we typically see over provisioning in ratio of 6:1 or 4:1 between I/O switch downlink ports and uplink ports. Depending on the I/O type Ethernet, FC, SAS etc. we may see larger over provisioning ratios.

5.6 Expected acceptance of MR-IOV

Precondition for MR-IOV to be accepted by the industry will be the availability of SR-IOV and MR-IOV capable devices. Nearly all PCIe controller vendors are developing SR-IOV devices. Looking at OS and VMM implementation effort to adopt MR-IOV, around 90% of the job will be SR-IOV related. Because nearly every PCIe controller vendor is going to provide SR-IOV capable devices in 2008 we expect that OS vendors like Microsoft, Sun or the Linux community will integrate necessary drivers in 2009.

We also think that several PCIe controller vendors will recognize the potential coming along with the MR-IOV approach. Even if vendors fear that the PCIe device savings due to the MR-IOV approach will harm profit, it might be realized also, that the MR-IOV approach will ensure a future for the PCIe endpoint business at all.

Finally, the amount of possible savings in power consumption and controller costs as well as an immediate integration of MR-IOV respective I/O management into data-center management tools will be key factors for a successful acceptance of MR-IOV by the industry.

6. Link Collection

http://www.pcisig.com/home
http://www.xensource.com/overview/Pages/overview.aspx
http://www.vmware.com/vinfrastructure/
http://www.intel.com/technology/platformtechnology/virtualization/index.htm

Disclaimer: All hardware and software names used are trade names or trademarks of their respective manufacturers.

Hardware Virtualization Exploiting Dynamically Reconfigurable Architectures

Jens Hagemeyer, Mario Porrmann
Heinz Nixdorf Institute
System and Circuit Technology
University of Paderborn, Germany
{jenze, porrmann}@hni.uni-paderborn.de

Markus Koester
Department of Computing
Imperial College London, United Kingdom
mkoester@doc.ic.ac.uk

Abstract

Integrating dynamically reconfigurable hardware into single or multi-computer environments poses specific requirements to the system designer and to the programmer, to facilitate high usability and efficiency of these heterogeneous systems. In this paper, virtualization methods are analyzed in respect to reconfigurable hardware. Techniques are described that combine various FPGAs to one large virtual FPGA including the possibility of dynamically reconfiguring only parts of the virtual FPGA as well as parts of single FPGAs. Partially changing the system architecture during run-time requires novel design flows and design methodologies. Appropriate approaches and implementations are sketched in this paper.

1 Introduction

In this paper we will analyze the benefits and the requirements when integrating dynamically reconfigurable hardware into today's computing environments with respect to virtualization. Operating system virtualization provides the illusion of many virtual machines (VM), each running its own operating system [1, 2]. Therefore, the available resources are shared between the various VMs, giving each operating system virtually exclusive access to the hardware infrastructure. Depending on the requirements of the OS and on the priority of the virtual machine (or the executed applications) different views concerning the available hardware can be offered to the operating system. This implies resources like CPU time, memory, and I/O (I/O virtualization). In addition to providing access to shared resources, virtualization is also used to combine distributed resources, like storage systems, to one large storage system that is easier to maintain and offers superior performance and reliability.

Reconfigurable hardware can be used to support the virtualization approaches described above. In [20] a Xilinx Virtex II Pro FPGA (field-programmable gate array), which includes reconfigurable resources and two PowerPC processors, is used to support I/O virtualization. Hardware and software mechanisms are analyzed, that enable concurrent direct network access by operating systems running within a virtual machine. The virtual machine monitor (VMM) offers each VM a virtual network interface that is multiplexed onto a physical network interface card. By utilizing additional hardware support for virtualization, the authors show a significant performance improvement compared to a pure software solution. With the growing need for virtualization solutions also the need for hardware support increases. Here, FPGAs can offer flexible solutions that can be easily adapted to new requirements. In today's scenarios FPGAs are typically used as a low-cost alternative to ASIC (application specific integrated circuit) implementations. But FPGAs offer a much higher potential – a lot of research in the reconfigurable computing area focuses on dynamic reconfiguration, i. e., the change of hardware (typically FPGA implementations) during run-time [3, 10, 19].

Dynamically reconfigurable hardware can efficiently support virtualization environments by adapting to changing operating conditions. Depending on the actual system state the same reconfigurable hardware can assist I/O or network virtualization at one time and accelerate an operating system or an application at another time. Additionally, due to the inherent parallelism of FPGAs, various hardware accelerators can operate concurrently without affecting one another. Therefore, the integration of FPGAs into today's computing environments can offer significant improvements in performance for virtualization, for the operating system, and for the application. Unfortunately, the integration of FPGAs comes with additional development cost; efficient implementations typically require an in-depth understanding of system architectures and of hardware development. This is especially true for the utilization of dynamic reconfiguration. Therefore, the main challenge is to efficiently utilize the dynamically reconfigurable resources and to offer easy to use frameworks that assist the developer

and that offer transparent access to the additional resources for the end-user application.

In this paper, we will focus on the efficient utilization of dynamically reconfigurable hardware. The new concepts can be used to improve the performance of virtualization environments as well as of applications – nevertheless they require additional virtualization effort. The rest of this paper is organized as follows. Section 2 gives an overview of hardware virtualization in respect to dynamically reconfigurable FPGAs. In Section 3 principles for the design of Multi-FPGA systems are discussed. This approach is detailed in Section 4 by adding the opportunity to partially reconfigure each FPGA in the system for optimum resource utilization. Finally, Section 5 discusses available high-level software tools and Section 6 concludes the paper.

2 Hardware Virtualization

In respect to dynamically reconfigurable hardware virtualization is typically associated with the execution of a large application on a hardware platform with insufficient available resources. Like virtual memory the virtual hardware is able to execute the application by time-multiplexing the available resources. The application is split into smaller parts that are executed sequentially on the hardware. In [17] this approach is referred to as *temporal partitioning*. Especially in embedded applications that have to cope with limited resources this is a very promising methodology to reduce cost and power.

Today's FPGAs have become quite large, offering the opportunity to operate different applications (hardware tasks) in parallel. These hardware tasks can be used to assist the virtualization environment, the operating system or as hardware accelerators for the end-user application. Therefore, an environment is required, that offers the opportunity to load hardware tasks on the FPGA, remove them and load new tasks during run-time. Like in software, the hardware tasks come with different requirements concerning reconfigurable resources (comparable to CPU usage), memory and I/O. Here, a hardware abstraction is required that manages the available resources – typically this is implemented in a *reconfiguration manager*. The reconfiguration manager decides where to place a hardware task on the FPGA, depending on the specific requirements and on the current device utilization. Comparable to I/O virtualization each hardware task has to be able to access the shared external interfaces of the FPGA. An important prerequisite for such an implementation is a homogeneous communication infrastructure on the FPGA, enabling communication independently from the location of the hardware task.

Although today's FPGAs are quite large, users often want to combine several devices to form one large virtual FPGA. Comparable to storage virtualization, concepts are required that efficiently map the application onto the FPGAs, taking into account, e. g., the diversity in communication bandwidth and latency between the devices. Ideally, the user does not recognize at all, where his application is located. It can be implemented completely in software, accelerated by one or more accelerators located on currently available reconfigurable resources in the system, or it is completely realized in hardware. For the virtualization environment this implies additional management effort, especially when dealing with hardware and software tasks with varying priority or with real-time requirements. In the following sections architectural aspects are detailed that are required to establish dynamically reconfigurable systems. A special focus is on the communication infrastructure – first for Multi-FPGA systems and subsequently for Single-FPGA environments.

3 Multi-FPGA Virtualization

A common scenario in hardware virtualization is the use of multiple FPGAs to execute one or more tasks. The main reasons for using virtualization are to map big tasks transparently to more than one FPGA and to eliminate the need to know which FPGAs in a given environment are actually used to execute a task. This chapter focuses on the foundation to build such a Multi-FPGA environment that can be used for Multi-FPGA virtualization later on. One of the most important aspects in the design of Multi-FPGA systems is communication. This includes the links between the FPGAs as well as the host coupling, witch defines how the FPGAs are connected to the rest of the system, e. g., to external CPUs in term of bandwidth and latency.

A typical mechanism to connect FPGAs to the host system is the utilization of a peripheral bus system like PCI-X or PCIe. However, it is also possible to connect an FPGA directly to the processor bus of the System, to use network connections like Ethernet, or even USB. All these possibilities have different assets and drawbacks. Figure 1 shows an example of a Multi-FPGA system, including FPGAs that are directly connected to the Front-Side-Bus of the Host (FSB, shown as red arrows in Figure 1). The direct connection to the FSB offers the highest bandwidth and lowest latency when communicating with other CPUs or with the system memory. However, due to space restrictions, typically only a single FPGA per CPU socket can be attached in such a manner. This FPGA is tightly coupled to the host system, but not to other FPGAs in the system.

FPGAs that are build on dedicated cards, e. g., a peripheral card attached to PCIe, can be connected to each other using dedicated, high-speed point-to-point connections because more than one FPGA (typical three to six) can be placed on the same board. However, the connection to the host system has to utilize a peripheral bus system like PCIe,

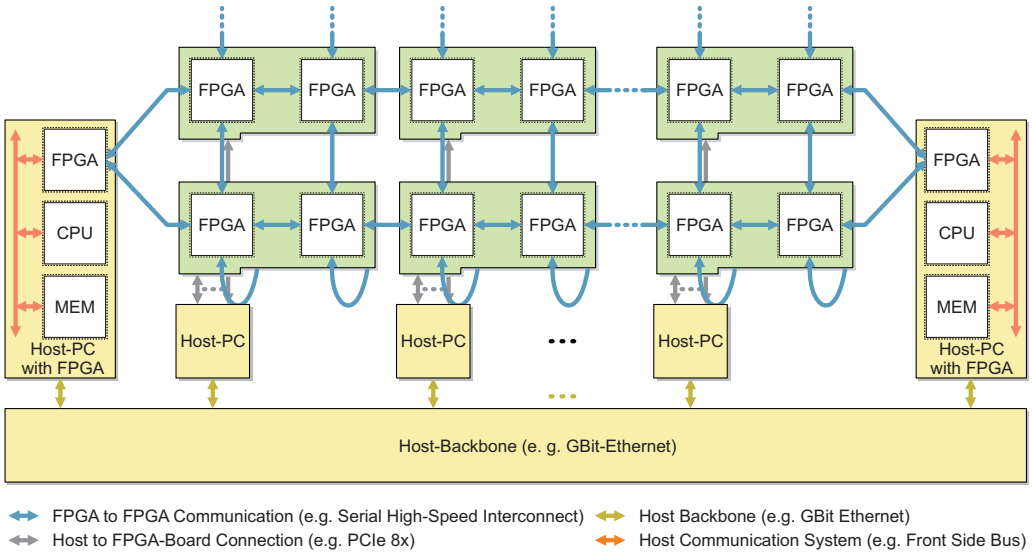


Figure 1. Overview over an example Multi-FPGA system

(shown as gray arrows in Figure 1), introducing additional delay and lower bandwidth compared to the FSB-based solution. To offer a solution for direct FPGA-to-FPGA communication between FPGAs connected to the processor bus and those located on peripheral cards, or between FPGAs located on different peripheral cards, high-speed serial connections can be used. High-speed serial interfaces enable copper-based high-speed connections for long range communication (typically 1 m), while normal LVDS connections are limited to short distances (typically up to 15 cm). High-speed serial links (shown as blue arrows in Figure 1) typically offer a medium bandwidth, low latency connection between FPGAs on different boards or platforms. In Table 1 an overview of typical connections for FPGA-to-FPGA and Host-to-FPGA communication is given. In this example, the high-speed serial interface is realized by four Rocket-IO interfaces, operating in parallel.

In the overview in Figure 1, a matrix topology was chosen as an example for the FPGA-to-FPGA communication. It is also possible to use other communication topologies, e. g., a 4D-hypercube, utilizing 16 FPGAs. More FPGAs can be added to the hypercube by utilizing additional con-

	Type	Bandwidth	Latency	Signaling	Duplex
FPGA-to-FPGA	On-PCB Connection (64 LVDS Connections)	80 GBit/s	10 ns	LVDS	Half-Duplex
	Rocket-IO (Link util. 4 x 6,5 GBit/s)	26 GBit/s	100 - 200 ns	LVDS/Aurora	Full-Duplex
Host-to-FPGA	FSB (FSB 1066 - Socket 604)	68 GBit/s	120 ns	AGTL+	Half-Duplex
	PCIe 8x (PCIe Version 1.1)	16 GBit/s	1-2 μs	LVDS/PCIe	Full-Duplex
	Ethernet (1000Base-T)	1 GBit/s	100 μs	PAM-5	Full-Duplex
	USB (USB 2.0 High-Speed)	0,48 GBit/s	1 ms	LVDS/USB	Half-Duplex

Table 1. Comparison of different FPGA-to-FPGA and Host-to-FPGA connection systems

nections, which are typically available between FPGAs located on the same board.

4 Single-FPGA Virtualization

In Multi-FPGA systems, a single FPGA is often considered as an atomic unit, that will be reconfigured completely if a reconfiguration takes place. A more generic solution is to configure only parts of an FPGA by using principles of partial reconfiguration. Using such techniques, the basic communication infrastructure for an application will stay the same, while hardware tasks, which are often called hardware modules in this context, will be placed somewhere in the cluster. To allow the placement of a given module at multiple positions on multiple FPGAs, homogeneity of the dynamically reconfigurable area is a major requirement. In this chapter, partitioning schemes for partially reconfigurable systems will be discussed and task placement techniques are introduced, which are required to place a hardware module in the environment. Finally, the special requirements of communication infrastructures in partially reconfigurable systems are discussed.

4.1 Partitioning

A partially reconfigurable system is composed of static components and dynamic components. Static components are the parts of the system that are always present, like the reconfiguration manager or the memory controller. Dynamic components are represented by hardware modules, e. g., accelerators for cryptography, for complex arithmetic, and for packet processing. Dynamic components are loaded at run-time by means of partial reconfiguration. Based on [25], a partially reconfigurable system can be partitioned as follows:

Base Region: The base region describes the area of the FPGA that is configured once at the initialization of the system. The configuration of the base region is not changed at run-time and can therefore be considered as pseudo-static. All static components of the system are located in the base region.

Partially Reconfigurable Region (PR Region): In contrast to the base region, the PR region is used for run-time reconfiguration. All dynamic system components are located in a PR region. A partially reconfigurable system can be composed of one or several separated PR regions.

Reconfigurable Tile: The PR region is composed of one or more individually reconfigurable tiles. A reconfigurable tile is the smallest partially reconfigurable unit. With respect to FPGAs it typically consists of several logic cells.

Partial Reconfiguration Module (PR Module): A partial reconfiguration module represents the implementation of a dynamic system component. It can be considered as a hardware module or hardware task in the context of partially reconfigurable architectures. PR modules can be placed and removed at run-time according to the needs of the application. The placement is done by reconfiguring suitable contiguous free tiles with the configuration data of the module.

In the simplest case, a single tile covers the whole area of the PR region. The corresponding modules can only consist of a single tile. As a consequence, the maximum number of modules that can be placed and executed at the same time is equal to the total number of PR regions. The advantage of this approach is that at run-time the placement of a module is done by simply selecting any suitable free PR region to place the module in. Implementations using this approach are presented, e. g., in [5, 24, 26]. However, the main disadvantage of this approach is the low resource utilization, since even small modules occupy a complete PR region. Furthermore, the size of a module is limited by the size of the tile.

To avoid these limitations, the PR region can be partitioned into multiple tiles as shown in Figure 2. A module is

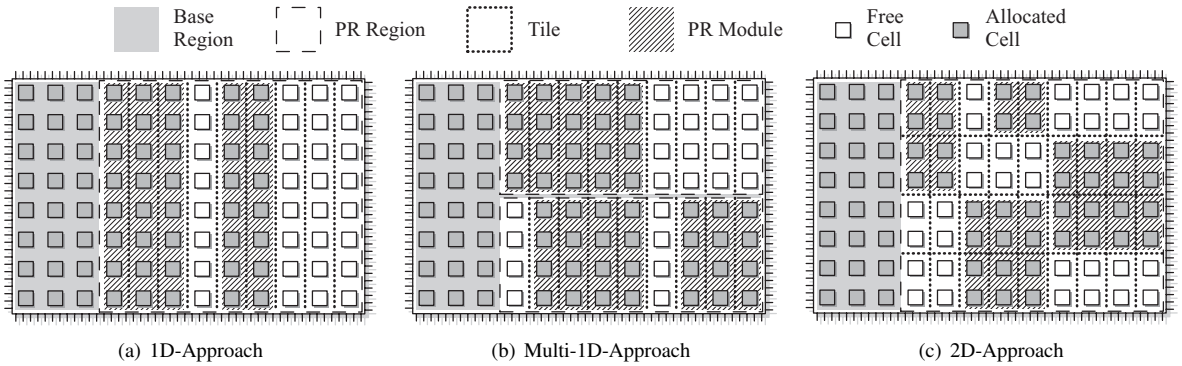


Figure 2. Overview of possible partitioning schemes for dynamically reconfigurable hardware.

no longer composed of a single tile, but of a group of contiguous tiles. Placing a module is equivalent to searching an area with as much contiguous free tiles as needed by the module (cf. Section 4.4). The number and sizes of the tiles and the arrangement within a PR region can be realized as described in the following approaches:

1D-Partitioning: In the 1D-approach shown in Figure 2(a) the height of the tiles is equal to the height of the PR region. The width of the tiles must be chosen according to the target FPGA. For Virtex-FPGAs, a commonly selected width is four CLB columns (cf. [12]). The tiles are arranged side-by-side and a requested module can be placed at positions with a sufficient number of free contiguous tiles.

Multi-1D-Partitioning: If the PR region is large, the aspect ratios of small modules in a 1D-partitioning can lead to an inefficient internal routing of the modules. Therefore, in the multi-1D-partitioning shown in Figure 2(b) the PR region is partitioned into equally sized subregions. Each subregion is again partitioned into multiple tiles as described earlier in the 1D-approach. The height of the tile corresponds to the height of a subregion. A requested module can be placed in one of the subregions at any position with a sufficient number of free contiguous tiles.

The multi-1D-partitioning is especially suitable for FPGAs with column-based partial reconfigurability, such as the Xilinx Virtex-4/5 devices, where the smallest partially reconfigurable unit is a configuration frame that consists of several vertically arranged logic cells. According to these devices, the height of a subregion is typically set to multiples of the height of a configuration frame.

2D-Partitioning: In the 2D-partitioning the PR region is partitioned into horizontally and vertically arranged tiles, as shown in Figure 2(c). A module is represented by a rectangularly-shaped group of tiles. In contrast to the multi-1D-partitioning, the height of a module is no longer defined by the height of the PR region. When generating a module, the area and the aspect ratio can be optimized according to the internal routing of the module. While the placement of a module is restricted to a one-dimensional search of free resources in the 1D-approach, the online-placement of a module in the 2D-approach requires a much more complex two-dimensional search.

In addition to the partitioning of the FPGA, the concept of partial reconfiguration requires a suitable communication infrastructure for interconnecting the PR modules and the base region. The communication infrastructure should not introduce any further heterogeneity in the system to maintain the flexibility of placement by preserving the number of feasible positions of the modules.

4.2 On-Chip Communication

When building an on-chip communication infrastructure for a partially reconfigurable system, it is necessary to define dedicated communication points, so that every partial reconfiguration module can connect to the communication infrastructure in the same way. To realize this, so called bus macros are used. Bus macros are pre-placed and pre-routed hard-macros (called Native Macro Component (NMC) in the Xilinx flow [26]) placed at a pre-defined positions.

Basically, two types of bus macros (aka hard macros) can be distinguished. The link macro (see Figure 3) acts only as a link between the base region (aka static region) and a reconfigurable tile. Alternatively, as depicted in Figure 4, a macro can be embedded in the reconfigurable area, connecting multiple reconfigurable tiles with each other and with the base region. Link macros are typically utilized in simple partially reconfigurable system, commonly used in embedded systems with a small number of reconfigurable tiles. Embedded macros, however, offer more flexibility for complex partially reconfigurable systems in large FPGAs or in Multi-FPGA systems.

An example implementation using embedded macros is shown in Figure 5. The figure shows the base region utilized by different static components, the communication macro, and two PR modules (hardware accelerators for AES-based decryption and floating-point multiplication) placed in the PR Region. While the use of an embedded macro has the disadvantages of a higher complexity during development, an important advantage of embedded macros is the possibility to put functionality that is required for the communication infrastructure into the embedded macro. An example is the integration of the address decoder into the communication infrastructure realized by an embedded macro. In contrast, typical link macros are just routing channels, wasting the occupied FPGA resources as route-through nodes. Furthermore, embedded macros can be used for every possible partitioning of a partially reconfigurable system, while link macros are restricted to setups where at least one side

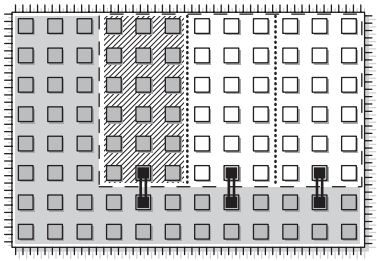


Figure 3. Example of a link macro implementation.

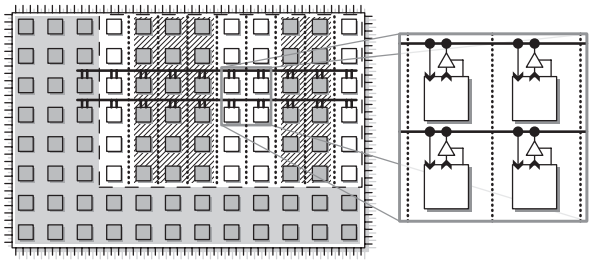


Figure 4. Example of an embedded macro implementation.

of a reconfigurable tile is adjacent to the base region. With respect to the homogeneity of the communication infrastructure, the use of embedded macros offers an advantage, because the routing of dedicated signals often destroys the homogeneity of a partially reconfigurable region when using link macros. The development of embedded macros is described in detail in [7] and [6].

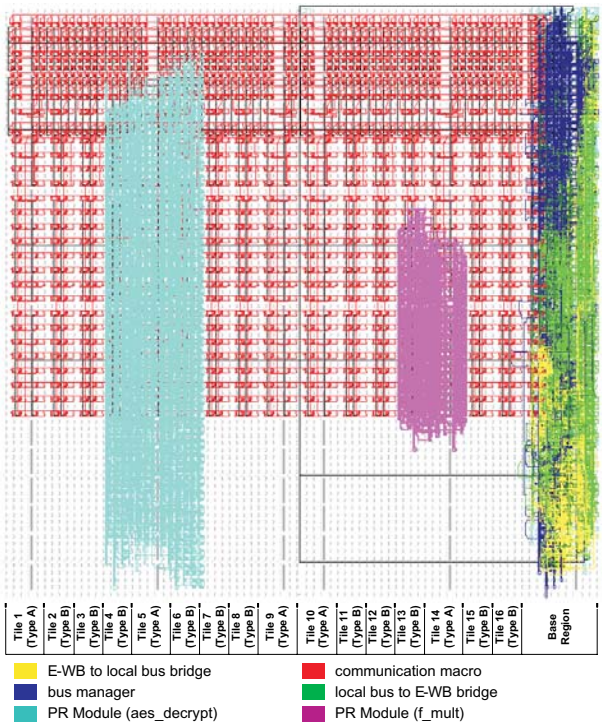


Figure 5. On-Chip communication infrastructure of a Virtex-2 XC2V4000-4 FPGA.

4.3 IO-Virtualization

The term "IO-Virtualization" refers to various methodologies in different contexts. In the context of a partially reconfigurable system on a single FPGA, some PR modules may require dedicated IO-connections, e. g., for control of external components that need direct control signals like analog-digital converters (ADCs). In general, a dedicated connection is a connection that is required exclusively by a PR module, and cannot be shared with other PR modules, like e. g. the data signals of a bus connection are shared between all PR modules. In the context of IO-Virtualization, the main problem is to ensure that a module connects to the correct IOs when it is placed somewhere in the reconfigurable area. As an additional constraint, the homogeneity of the reconfigurable area must be preserved. The problem can be generalized to a dedicated signal problem, as dedicated signals not only exist for IO-connections, but in almost every communication infrastructure, that require dedicated control signals like enable signals. One solution to implement a dedicated signal in a homogeneous communication structure is to use some registers, which never change their initial value, in combination with some decoding logic. Details to this a approach and alternatives are described in [7] and [8].

Another, more specific solution is the use of a multiplexer block in the base region to connect a partial reconfigurable module with the correct IO. For an efficient implementation of the multiplexer block, it is possible to use techniques exploiting partial reconfiguration as well. As the connections in the multiplexer block just need to be changed when a reconfiguration takes place, it is possible to implement the multiplexer block itself as a (very small) dedicated reconfigurable module and add the necessary configuration information to the reconfiguration bitstream. Using this approach, there is no additional routing delay introduced by the multiplexer block, because only routing resources of the FPGA are used for implementation of the multiplexer block. No additional logic resources are required.

In another context, IO-Virtualization refers to connections between FPGAs that are used to create a communication infrastructure to build one big, virtual FPGA. As already mentioned in Section 3, different types of connections can be used to implement such communication structures. The communication links can be used exclusively by one hardware task of the complete design mapped to the big, virtual FPGA, or, as in most cases, shared (e. g., time-multiplexed) by many hardware tasks. One principle difference between the different types of links is the exact knowledge about the link delay at design time. Some examples of possible connections are:

Direct Connection: Using a direct connection, e. g., an on-PCB connection, is the simplest case of inter-FPGA connec-

tion. The delay of the line is determined only by the number of pipeline stages in the FPGAs. This simplest case can be handled by semi-automatic Multi-FPGA partitioning tools like Synplicity Certify [[22]].

Serial High-Speed Connection: A serial high-speed connection has no deterministic delay. Since the effective delay of a high-speed serial connection depends on the phase relationship and the lock behavior of the integrated PLLs (Phase Locked Loop), it will vary between every new startup synchronization of the high-speed serial connection, which is required, e. g., at system startup or after reset. It is possible to implement additional buffer mechanisms to circumvent those problems and to achieve a deterministic delay at design time. However, this will slightly increase the total delay of the connection. Another possibility is to use a data-driven architecture, which does not need deterministic delays at design time.

Bus Connection: A bus connection in general cannot offer a deterministic delay, because the arbitration will cause a delay that depends on the current bus load. However, when implementing a special protocol, e. g., a time-slot based protocol on a bus-like communication infrastructure, it is possible to achieve a deterministic communication delay.

4.4 Task Placement

The partitioning of the FPGA (cf. Section 4.1) and the development of a suitable on-chip communication infrastructure (cf. Section 4.2) are important steps in the design of a partially reconfigurable system. One of the major challenges with respect to the run-time behavior of a partially reconfigurable system, is the placement of hardware tasks in the partial reconfiguration region, which has some similarities to the memory allocation problem of software tasks in an operating system. Essentially, the problem is about finding a reasonable amount of free resources to map the task to. Software tasks utilize the resource memory to store control and application data, while hardware tasks utilize the resource memory to store configuration data. But there are some differences that are briefly described in the following.

One major difference is the fact that hardware tasks cannot be subdivided into smaller fragments. The resources need to be available in one contiguous block. Therefore, a concept equivalent to virtual memory cannot be applied. Although the FPGA itself is a fine-grained reconfigurable architecture, the reconfigurable tiles can be considered as coarse-grained partially reconfigurable units. The upper bound for the number of feasible positions for a PR module is the number reconfigurable tiles. Moreover, the PR region can be composed of various types of tiles, e. g., tiles consisting of logic cells only and tiles consisting of logic cells and Block RAM (cf. Figure 5). The diversity of tiles further restricts the placement of hardware tasks to those positions, where the same arrangement of the required types of tiles occurs. Additionally, the placement may be subject to application-dependent constraints. E. g., if the requested task needs to communicate to one or more functional dependent tasks without exceeding a certain latency, then the requested task needs to be placed within a given range around its dependent tasks to meet the latency constraint. Especially, in application with data streams, such as video filters, this latency constraint might be relevant for the placement.

Depending on the given reconfigurable region, tasks can also be placed in two dimensions, while the memory allocation problem in context with software tasks has a one-dimensional search space. But the coarse-grained partial reconfigurability and the additional application-dependent constraints restrict the placement of a task to a few feasible positions. The search space for the placement of a hardware task is therefore significantly smaller than compared to the memory allocation problem in context with software tasks. In the example shown in Figure 5 the partially reconfigurable region consists of 16 tiles only.

In general, the placement of hardware tasks can be classified into online-placement and offline-placement. Offline-placement denotes the case that the starting and execution time of each hardware task is known at design-time. Here, the placement problem is equivalent to a three-dimensional binpacking problem that can be solved offline by integer linear programming as described, e. g., in [4]. The area of a hardware task takes two dimensions, while the third dimension denotes the execution time. The execution time of a task can be derived, if the input data size and the processing time is known. Possible examples are filters for digital images of a given size.

The placement needs to be done online, if the starting times of the tasks, i. e., the moments the tasks are requested to be placed, are unknown at design-time. In particular, the starting times of the tasks are unknown in applications with event-driven reconfiguration. An example for an event-driven reconfiguration is a task that is requested to be placed after another task with an unknown execution time is finished, or a task that is requested to be placed on demand by the user. In any application with event-driven reconfiguration, the placement is an online-optimization problem, which basically depends on the currently configured hardware tasks. The type of the online-optimization problem is related to the type of the hardware tasks. If the execution times of the tasks are given, the placement is equivalent to a three-dimensional online binpacking problem and can be solved by heuristics as described in [21]. If the execution times of the tasks are unknown, the problem size decreases to a two-dimensional online binpacking problem. A suitable solution for this kind of task placement is presented

in [13]. This work also considers the placement for heterogeneous reconfigurable architectures with various types of reconfigurable tiles.

If a hardware task is requested to be placed, but there are not enough contiguous free resources to accommodate the requested task, then there are different possibilities to handle this unplaceable task. In the simplest case, the task placement can be rejected. This is suitable for systems, where the task can be alternatively executed as a software task, such as described in [18]. Another possibility is to delay the placement of the task until another task has finished its execution and can be removed to free a reasonable amount of resources. This kind of delayed placement is discussed in [21]. Instead of waiting for the end of an execution, the currently configured tasks can also be rearranged aiming to create a block of contiguous free resources to accommodate the requested task. This concept is known as defragmentation. The required mechanisms to relocate a hardware task at run-time and defragmentation algorithms aiming to minimize the reconfiguration overhead are described in [14].

5 Design Flow

To use a system like the one displayed in Figure 1, an appropriate design flow is needed. This design flow should be able to deal with the mapping of one application to multiple FPGAs as well as with the special issues arising from the use of partial reconfiguration. In Figure 6, the abstract view of a typical design flow for dynamically reconfigurable hardware/software systems is shown. Basically, there are two starting points in the design flow. One is the design of the basic hardware architecture that runs the hardware tasks, and, as its counterpart, the design of the operating system. The second starting point is the application or a set of applications that are mapped on the virtual FPGA, on the CPU, or on both.

During the mapping of the application, the first step is HW/SW partitioning. Depending on the available specification of the application, coding in hardware description languages like VHDL or Verilog may be required for the hardware tasks and, e. g., C-code is generated for the software tasks. Additionally, novel tools can be utilized that enable C to hardware compilation. Depending on the used C to hardware compiler, it may be necessary to use annotated C code like Handle-C or System-C. Alternatively, VHDL compilers like the Mitrion SDK [16] or Impulse C [9] can be used with native C code. However, the efficiency of the retrieved hardware implementation (in terms of performance and resource requirements) is very application dependent, when using C to hardware compilers. Therefore, the developer has to trade off design-time against performance. As an alternative, the developer can start with domain-specific tools

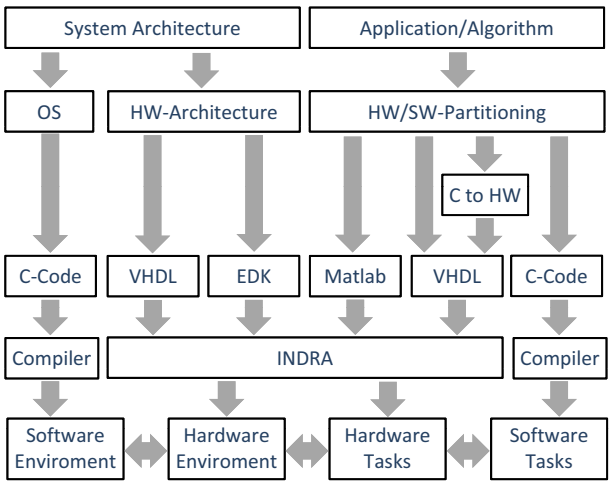


Figure 6. Design flow for dynamically reconfigurable hardware/software systems

like MATLAB/Simulink. Synplify DSP [23] or Xilinx System Generator [27] extend the established framework and are able to generate hardware descriptions based on abstract simulation models. These tools rely on optimized libraries with generic elements that are parametrized during hardware generation.

The basic hardware architecture needed for a partially reconfigurable system, including the interfaces to other FPGAs, the base system, and the on-chip communication infrastructure is typically described in VHDL. To shorten the development time, a design tool for embedded systems, like the Xilinx Embedded Development Kit (EDK), can be used. The software part of the system, i. e., the software tasks as well as the operating system, are compiled using appropriate cross-compilers for the processor-types in the system. All VHDL descriptions and netlists are handled by the INDRA design flow, which is described next.

The INDRA design flow (integrated design flow for reconfigurable architectures) is depicted in Figure 7. The description of the base system of the design, supplied by the HW-architecture development process, is used as the static part of the system. The area that is used for the static components is also referred to as the base region. The given information of the system components is used in the layout and floorplanning step to determine the required resources. The number of resources for the static components does not change at run-time. Thus, it can be derived directly from the netlists or from the synthesis of the HDL specifications. In contrast to the static components the dynamic components are allocated at run-time. Since dynamic components share the resources of the partially reconfigurable region, the number of necessary resources depends on module pa-

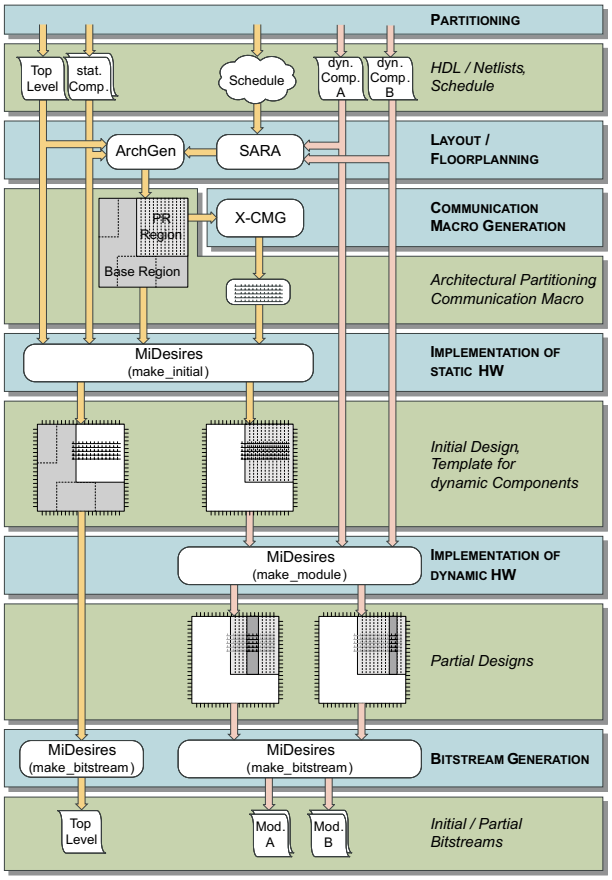


Figure 7. Overview of the INDRA design flow

rameters, such as the sizes and the execution times of the modules, and on how many modules are allocated at the same time. For a given schedule, the size of the partially reconfigurable region can be tested by using the simulation framework for reconfigurable architectures (SARA) [10]. Besides the partitioning, SARA can be used to select a placement algorithm for finding positions of the PR modules at run-time. Based on the simulation results a suitable FPGA and placement algorithm is selected. In the architecture generation (ArchGen) step the system is partitioned and the areas for the base region and for the partially reconfigurable region are defined. Synthesis of the base region and of the PR modules is performed based on the system partitioning. Depending on the size of the modules, which is obtained from synthesis estimation, and on the inherent heterogeneity of the FPGA, INDRA automatically determines the steps required for the synthesis of the PR modules.

After the architecture generation step, all requirements for the needed communication infrastructure are specified. The communication infrastructure for a dynamically reconfigurable system, which supports flexible module placement and module relocation at run-time, requires being homogeneous. Homogeneity implies that the individually reconfigurable tiles are connected by the same routing resources. Thus, modules cannot only be placed at one dedicated position, but at any position with sufficient free contiguous tiles. To generate a homogeneous communication infrastructure, X-CMG [7] has been developed. X-CMG features a multilevel, primitive-based communication macro generation approach. To build a homogeneous communication infrastructure, X-CMG offers primitives for different classes of signals. The primitives can be used to implement shared signals and dedicated signals as discussed in [8]. Shared signals are used to transmit data and address information, while control and arbitration is realized with dedicated signals.

All previously described steps are managed by the system designer and are part of the INDRA front-end. The following steps aim at generating the bitstreams for the base region and the partially reconfigurable region. These steps are realized by MiDesires (Module implementation design flow for reconfigurable systems), which describes the back-end of INDRA. It is based on the current Xilinx JTRS flow as presented in [15]. MiDesires is not only an interface to the Xilinx tools, as it provides additional features like saving and loading the current state of the design flow. Additionally, it automates required steps during the module synthesis such as the adaptation of UCF files and resolving of dependencies. Furthermore, it directly changes parts of the Xilinx design flow by modifying internal PAR settings accordingly and integrates mechanisms for module relocation as presented in [11].

6 Conclusions

In this paper, various alternatives for the integration of dynamically reconfigurable hardware into today's computing environments have been presented. On the one hand, reconfigurable hardware can be efficiently utilized to support IO virtualization or operating system virtualization. On the other hand, virtualization methodologies are required to enable or enhance the capabilities of dynamically reconfigurable hardware. Virtualization can be applied on single FPGAs, e. g., for resource sharing and time-multiplexing. But it can also be used to combine multiple devices to one large virtual FPGA, e. g., for ASIC prototyping.

In order to apply the proposed concepts, several requirements need to be met at design-time and at run-time. In the single-FPGA approach the device is partitioned into a static and a dynamic part. A homogeneous communication infrastructure is presented, that allows most flexible placement. The proposed concepts can be applied to any FPGA supporting partial reconfiguration. Currently, these are the Xilinx devices of all Virtex families. At run-time, the tasks

are placed depending on the actual system state and on the system requirements. In multi-FPGA environments, special care has to be taken when designing the communication infrastructure between the FPGAs as well as between the FPGAs and the host systems. Bandwidth and latencies vary by several orders of magnitude depending on the implemented infrastructure. At run-time, an efficient partitioning has to assure the efficient utilization of the proposed heterogeneous system. A combination of many, to some extend proprietary tools is required to design, program and operate these systems. The proposed design-flow INDRA serves as a starting point for the integration of the complete workflow into one framework.

References

[1] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proc. of the 12th Int. Conf. on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proc. of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[3] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.

[4] K. Danne and S. Stühmeier. Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants. In *Proc. of Int. Embedded Systems Symposium 2005 (IESS)*, 15–17 2005.

[5] F. Ferrandi, M. D. Santambrogio, and D. Sciuto. A design methodology for dynamic reconfiguration: The caronte architecture. In *Proc. of the 19th Int. Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2005.

[6] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann. A Design Methhodology for Communication Infrastructures on partially reconfigurable FPGAs. In *17th Int. Conf. on Field Programmable Logic and Applications (FPL2007)*, pages 331–338, 27 - 29 Aug. 2007.

[7] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann. Design of homogeneous communication infrastructures for partially reconfigurable fpgas. In *Proc. of the Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA '07)*. CSREA Press, 2007.

[8] J. Hagemeyer, B. Kettelhoit, and M. Porrmann. Dedicated module access in dynamically reconfigurable systems. In *Proc. of the 20th Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[9] Impulse Accelerated Technologies. Impulse CoDeveloper C-to-FPGA Tools, 2007.

[10] H. Kalte, B. Kettelhoit, M. Koester, M. Porrmann, and U. Rückert. A system approach for partially reconfigurable architectures. *Int. Journal of Embedded Systems*, 1:274–290, 2005.

[11] H. Kalte, G. Lee, M. Porrmann, and U. Rückert. REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems. In *Proc. of the 19th Int. Parallel and Distributed Processing Symposium*, 2005.

[12] H. Kalte, M. Porrmann, and U. Rückert. System-on-programmable-chip approach enabling online fine-grained 1D-placement. In *11th Reconfigurable Architectures Workshop*, 2004.

[13] M. Koester, H. Kalte, and M. Porrmann. Task placement for heterogeneous reconfigurable architectures. In *Proc. of the IEEE 2005 Conf. on Field-Programmable Technology (FPT'05)*, pages 43–50. IEEE Computer Society, 2005.

[14] M. Koester, H. Kalte, M. Porrmann, and U. Rückert. Defragmentation algorithms for partially reconfigurable hardware. *VLSI-SoC: From Systems to Silicon, IFIP Int. Federation for Information Processing Series*, pages 41–53, 2007.

[15] P. Lysaght, B. Blodget, J. Mason, B. Bridgford, and J. Young. Enhanced Architectures, Design Methodologies and CAD Tools for dynamic reconfiguration of XILINX FPGAs. In *16th Int. Conf. on Field Programmable Logic and Applications (FPL2006)*, pages 12–17, 2006.

[16] Mitrionics Inc. Mitrion Product Brief, 2007.

[17] C. Plessl and M. Platzner. Virtualization of hardware - introduction and survey. In *Proc. of the Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA '04)*, pages 63–69. CSREA Press, 2004.

[18] F. J. Rammig, M. Götz, T. Heimfarth, P. Janacik, and S. Oberthür. Real-time operating systems for self-coordinating embedded systems. In *Proc. of the 9th IEEE Int. Symposium on Object and component-oriented Real-time distributed Computing (ISORC 2006)*, Gyeongju, Korea, 24 - 26 Apr. 2006.

[19] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, and T. Becker. Modular partial reconfiguration in Virtex FPGAs. In *15th Int. Conf. on Field Programmable Logic and Applications (FPL2005)*, pages 211–216, 24 - 26 Aug. 2005.

[20] J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, W. Zwaenepoel, and P. Willmann. Concurrent direct network access for virtual machine monitors. In *HPCA '07: Proc. of the 2007 IEEE 13th Int. Symposium on High Performance Computer Architecture*, pages 306–317, Washington, DC, USA, 2007. IEEE Computer Society.

[21] C. Steiger, H. Walder, and M. Platzner. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Trans. Computers*, 53(11):1393–1407, 2004.

[22] Synplicity Inc. Certify Datasheet, 2007.

[23] Synplicity Inc. Synplify DSP Datasheet, 2007.

[24] M. Ullmann, M. Hübner, B. Grimm, and J. Becker. An FPGA run-time system for dynamical on-demand reconfiguration. In *Proc. of the 18th Int. Parallel and Distributed Processing Symposium*, 2004.

[25] Xilinx. Early access partial reconfiguration user guide. In *UG208*, 2006.

[26] Xilinx Inc. Application notes 290. Two flows for partial re-configuration: Module based or small bit manipulations, 2002.

[27] Xilinx Inc. System Generator for DSP User Guide, 2007.

Cost Saving and Flexibility - Virtualization is changing the IT landscape

Christoph Dobroschke
Advanced Micro Devices (UK) Ltd
Christoph.Dobroschke@amd.com

Abstract

As organizations increase their reliance on information technology to meet critical business objectives, IT must continually deploy new computers to implement, maintain and grow services rapidly. Unfortunately, the traditional approach to provisioning x86 systems has given rise to server sprawl and a number of significant challenges, rising costs, poor return on investment, increased management complexity and reduced efficiency. This situation has put great pressure on IT budgets and schedules, leading many organizations to search for a solution that can help them sustain rapid growth while reducing operating costs. Virtualization technology is now emerging as a key tool for several of these issues. This presentation will give an overview of the current IT landscape, business segments where virtualization is already rolling out and an outlook of where we anticipate virtualization technology playing a major role in the future. It will also discuss current and upcoming technologies enabling various forms of virtualization:

1. Current IT challenges
2. How virtualization technology can help ease the pain
3. Market segments that are driving the adoption of virtualization
4. Example business cases
5. Hardware extensions to support virtualization
6. Trends and upcoming technologies

Netzwerk-Virtualisierung

A Network Virtualisation Concept based on the Ambient Networks SATO System

M. Stiemerling[+*], X. Fu[*], and M. Brunner[+]
[+]*NEC Europe Ltd., NEC Laboratories Europe*
[*]*University of Göttingen, Institute for Computer Science*
{stiemerling, fu}@cs.uni-goettingen.de, brunner@nw.neclab.eu

Abstract

Network virtualization can be one way of fixing the shortcomings of today's Internet but also open the venue for new, unforeseen applications. In this extended abstract, we present a novel approach for network virtualisation based on the Service-Aware Transport Overlay (SATO) concept of Ambient Networks. SATOs introduce on-demand overlay creation and new interfaces to ease applications to use overlays.

1 Introduction

Internet applications often assume the end-to-end connectivity for their operations. While this might be true in some areas of the Internet, it is not true for the overall Internet. Many applications today need to take care of network-specific behaviour, such as broken end-to-end connectivity (e.g. NAT), directionality of communication (e.g. IP firewall), congestion control/avoidance (important for real time applications), and different network layer addressing schemes (IPv4 vs. IPv6). This challenges applications in multiple ways, for example, in terms of interface usage (necessity to deal with different TCP/IP sockets), taking care of middlebox behaviour. The result is on one hand complex applications that are error prone with respect to a variety of network behaviour and determination of network behaviour. On the other hand, some applications possibly cannot be deployed or are delayed in deployment, as the due to the constraints of the underlying network. For instance, deployment of SIP-based services took a long time due to the network limitations in terms of NATs and the "unusual" behaviour of unbundled SIP signalling and the transport of media.

One way to address this issue is aiming at a clean slate Internet [11], believing in a forthcoming evolution of the Internet towards a clean end-to-end architecture (e.g. IPv6, [12]). Another approach is to introduce short-term patches, for example, revamping every real time application with ICE NAT traversal techniques [13]. Also a typical short-term response of application developers is to create their own specific overlay system. These overlay systems are made for a single, or very limited range, purpose and not suitable for other purposes. Examples for this are Skype for voice/video calls [14], Joost for IPTV [15], and MBONE for multicast [16]. However, we do not believe that any of these strategies is the key to solve application's needs for an easier to use of the current Internet and beyond. A clean slate of the Internet is not possible as the installed base of IP devices is too large and the Internet community always runs on the assumption that there is no flag day for changing fundamental parts anymore. The Internet evolves on the assumption of gradually deployment of new functions, e.g., IPv6 is available in many operating systems but not in the Internet core.

All these increase the need for applications to be aware of the network behaviour and to deal with the network mainly on their own. Introducing new patches are usually not simplifying the way in which a network can be used by applications. For instance, introducing IPv6 required changing applications (e.g., a new socket interface) and operational considerations when using IPv6 (e.g. DNS returns an IPv6 address but the querying host is IPv4 only). [2] refers to this process of an evolving the Internet with more features and patches as the ossification of the Internet.

One way to offload applications from this burden is to introduce the concept of network virtualization, as proposed in [1], i.e., offering an abstraction to higher layers from the real network. [2] describes adding network processing functions, based router hardware add-ons, to boost network virtualization and [3] describes the software-base X-Bone concepts for deploying virtual networks.

Based on these ideas of dynamically deploying overlays and network side processing support, we propose a new approach combining these two aspects. This overlay technique includes usage of network side processing elements system beyond pure routing and automatic adaptation to network changes. The proposed system introduces a new network interface between overlay nodes to control the overlay and argues for a new system interface between applications and the host network stack.

The work presented is based on the Service-Aware Transport Overlays (SATO, [6]) of the Ambient Networks Phase 2 project [17].

2 Service-Aware Transport Overlays

2.1 Overview

The main objective of the Service-aware Adaptive Transport Overlay (SATO) concept, proposed in [6] and extended in [7], is to design a generalised overlay system structure, which is able to realise overlay networks on demand. More specifically, it introduces a uniform overlay infrastructure to support multiple applications and to provide them with useful functionalities realised inside the network paths, thus providing an abstraction from the underlying network. Part of the SATO system is the Ambient Service Interface (ASI), which provides an abstract network interface to applications.

SATO aims to provide a flexible and customisable transport services to the application layer by using overlay networks on top of any type of transport or network layer connectivity. The individual overlay instances, i.e., SATOs, enable the flexible configuration of virtual networks consisting of SATO Overlay Nodes (SON). The overlay network topologies are responding to the application requirements and can enable point-to-point, point-to-multipoint and multipoint-to-multipoint services. SATOs are created upon application request and application needs, they are set-up on and torn down on demand. The SATO concept allows the transparent inclusion of network-side data processing elements (SATO Ports) in the end-to-end transport path between peers. These SATO-Ports can provide network side processing functions, such as, but not limited to, overlay routing, media adaptation, etc. To accommodate changes in the underlying network, SATOs adapt as a consequence of SON joining or leaving the virtual network or due to changes in the network environment or conditions.

For each requested service, a new SATO is created. This SATO is self-contained and logically does not interfere with any other existing or future SATOs. Service requests must be issued per SATO and they can contain necessary parameters, such as required bandwidth, delay, jitter, etc, or just express a rough service class, e.g., signalling or real-time media.

2.2 Architecture

The core of the SATO system are the SATO Overlay Nodes (SON) that participate in the SATO system by providing their network and computing resources to other nodes. A SON can be any kind of Internet device, starting from a mobile phone to fixed PC in an office. A SON can be part of any number of SATO instances, providing different services to different applications.

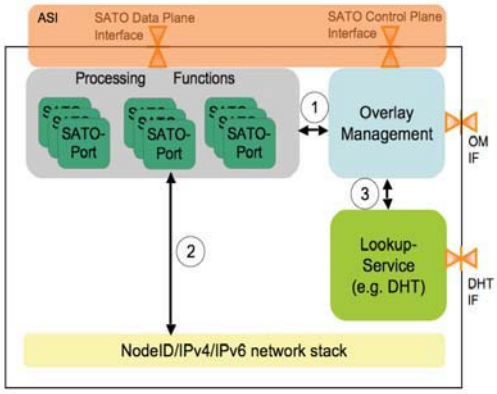


Figure 1 SATO Overlay Node Design

Within a SON there are different building blocks, as shown in Figure 1. The Overlay Management (OM) is in charge of operating the SATO system, i.e., receiving requests from applications for setup through the control plane interface, maintaining existing SATOs and tearing down of SATO. The OM is fully distributed amongst participating overlay nodes. Applications can request a SATO service through the SATO control plane interface. The OM interface (OM IF) is used by the OM to communicate with other OMs on other nodes. The OM of the SATO system is aware of the service requested by the application and about the network conditions in which the service is operated (the so-called service logic).

Each SON is part of the SATO lookup service, which is typically a distributed hash table (DHT), but not limited to, e.g., a centralised lookup service approach could also be chosen for a deployment of SATO. The lookup service and the OM communicate with each other using the interface 3. This lookup service is used to store information required for operating the SATO system and information required for running the applications. The OM is primarily using the lookup service to store information about available processing functions, such as packet relays, and their location in the network, i.e., the IP addresses/NodeIDs.

The processing function block that hosts the SATO-Ports provides the real functionality of the SATO system. A SATO-Port provides an arbitrary processing functionality, for example, MPEG transcoders, voice-codec transcoders, peer-to-peer SIP proxies, or overlay routers (called virtual router in [2]). SATO-Ports have one external interface to the application, which is the SATO data plane interface in Figure 1. Interface 2, called SATO network interface, is connecting the SATO-Ports to the network stack of the SON and thus to other SATO-Ports. The control of the SATO-Port by the OM is provided by interface 1. The SATO system

is able to include SATO-Ports on the fly into existing SATO instances, or when setting up a new SATO. Whenever a processing function is needed (this is decided by the OM when constructing or adapting the overlay) a search in the lookup service is performed for the type of SATO-Port required.

3 Network Virtualization

Network virtualization follows the same principle as any known computer virtualization technology, such as virtual memory, virtual hard disk, virtual screens, etc. It is always an abstraction from a real resource, hiding the underlying complexity (cf. [1]). But even if virtualization is an integral part of today's computing technology (which is close to communication technology), it is still lacking a counterpart in the Internet architecture. Current proposals discuss possible virtualization techniques (e.g. [3]) on top if the Internet, but do not consider how this can be integrated in the architecture.

Network virtualization as an integral part of the future Internet has been proposed by, for instance by GENI [8]and [9] on the architectural level. We propose to add overlay techniques as means of network virtualization to the Internet architecture. The overlay system is not introduced together with one new IP host to IP host interface and an enhanced application to network stack interface.

3.1 SATO for Network Virtualization

Section 2.1 defines SATO as a generic overlay system that is not bound to a specific networking application, but generic enough to be reusable by many networking applications. This generic approach also allows building a network virtualization system. The very base of the SATO system is the NodeID architecture [4], which extends the Host Identity Protocol [5] by adding IPv4 to IPv6 support and a new routing architecture based on cryptographic identifiers. The NodeID is used as it provides cryptographic node identifiers, by-default encryption between nodes and an abstraction from the IP level. This first level of abstraction is shown as *NodeID level* in Figure 2. The *NodeID level* hides underlying Network Address Translator (NAT) middleboxes and ensure virtual links from peer-to-peer, or peer to SATOPort.

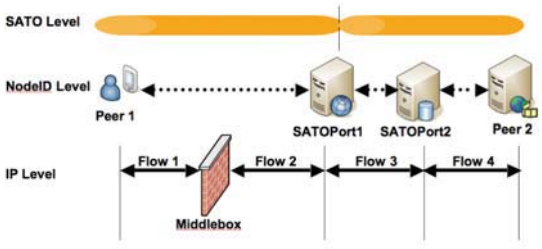


Figure 2: Abstraction Level of SATOs

The next level of abstraction is the SATO level in Figure 2. The SATO level is abstracting from the NodeID level to built the overlay topology. The SATO level is the only visible layer to applications requesting a virtual network. This virtual network, for example, can provide an IPv6 network on top of an IPv4 network. Applications using this virtual network would only be able to see a subset of the SATOPorts (a virtual IPv6 router in this case) as part of the network infrastructure. Other SATOPorts (SATOPort 2, e.g. traffic shaper, in Figure 2) stay invisible to the applications.

3.2 Network Interface

The deployment of overlay networks used for network virtualization, for example the 6BONE [10], requires manual configuration and network management. Adding now network virtualization to the Internet raises the need for coordinating these efforts. Network virtualization would be an on demand service, such as TCP connections are today, but with more network elements involved (e.g., virtual routers). This demands a control instance between participating nodes.

The SATO system is already offering on-demand network virtualization, together with a network control interface, i.e., the overlay management interface (OM IF, cf. Section 2.2). This interface allows coordination amongst participating nodes of the overlays and control of the overlays and can be used for this purpose.

3.3 Host Interface

The SATO system has the notion of the Ambient Service Interface (ASI). The intention of the ASI is to offer a more enhanced application to network stack interface, i.e., host interface. However, the ASI as such is currently not well defined and underspecified. We propose to extend the scope of the ASI to support network virtualization techniques on the control and data plane. This would allow applications to directly use a virtual network without the requirement for implementation hacks, such as for example, intercepting packets with firewall rules or kernel drivers.

4 Conclusion

This extended abstract sketches the vision of network virtualization as service-specific on-demand overlay networks with network side processing elements. We have proposed the SATO system as one possible exploration tool in the area of network virtualization but we are fully aware that this is not the ultimate solution to general field of network virtualization. The SATO system, with NodeID, the new network interface, and the enhanced host interface is work in progress. The proposed approach is intended to raise further questions about the direction of network virtualization as part of Internet architecture and if overlays are the right answer to this challenge.

5 References

[1] L. Peterson, S. Shenker and J. Turner, Overcoming the Internet Impasse through Virtualization, Hotnets 2004.

[2] J. Turner and D. Taylor, Diversifying the Internet, Proceedings of Globecom 2005.

[3] J. Touch, Y. Wang, L. Eggert, , and G. Finn, A Virtual Internet Architecture, ACM SIGCOMM Workshop on Future Directions in Network Architecture, 2003. Karlsruhe, Germany.

[4] A Node Identity Internetworking Architecture. Bengt Ahlgren, Jari Arkko, Lars Eggert and Jarno Rajahalme. Proc. 9th IEEE Global Internet Symposium (GI 2006), Barcelona, Spain, April 28-29, 2006.

[5] R. Moskowitz and P. Nikander, Host Identity Protocol (HIP) Architecture, RFC 4423, May 2006.

[6] Ambient Networks Deliverable: “System Design of SATO & ASI”, website http://www.ambient-networks.org/Files/deliverables/D12-F.1_PU.pdf, December 2007.

[7] M. Stiemerling and M. Brunner, A Peer-to-Peer SIP System based on Service-Aware Transport Overlays, VoIP Themenheft, Praxis in der Kommunikationstechnik, October 2007.

[8] Larry Peterson, Tom Anderson, Dan Blumenthal, Dean Casey, David Clark, Deborah Estrin, Joe Evans, Dipankar Raychaudhuri, Mike Reiter, Jennifer Rexford, Scott Shenker, and John Wroclawski, "GENI design principles," in IEEE Computer, September 2006.

[9] Stephan Baucke, Ibtissam El Khayat, Ralf Keller, Norbert Niebert, Flexible Architecture for the Future Internet, Joint EuroFGI and ITG Workshop on "Visions of Future Generation Networks", Würzburg, Germany, July 2007.

[10] http://go6.net/ipv6-6bone/6bone_hookup.html, December 2007.

[11] http://cleanslate.stanford.edu/, December 2007.

[12] S. Deering and R. Hinden, Internet Protocol, Version 6, RFC 2460, December 1998.

[13] J. Rosenberg, Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, Internet draft (work in progress) draft-ietf-mmusic-ice-19, October 2007.

[14] Skype, http://www.skype.org, December 2007.

[15] Joost, http://www.joost.org, December 2007.

[16] MBONE, http://www-mice.cs.ucl.ac.uk/multimedia/projects/mice/mbone_review.html, December 2007.

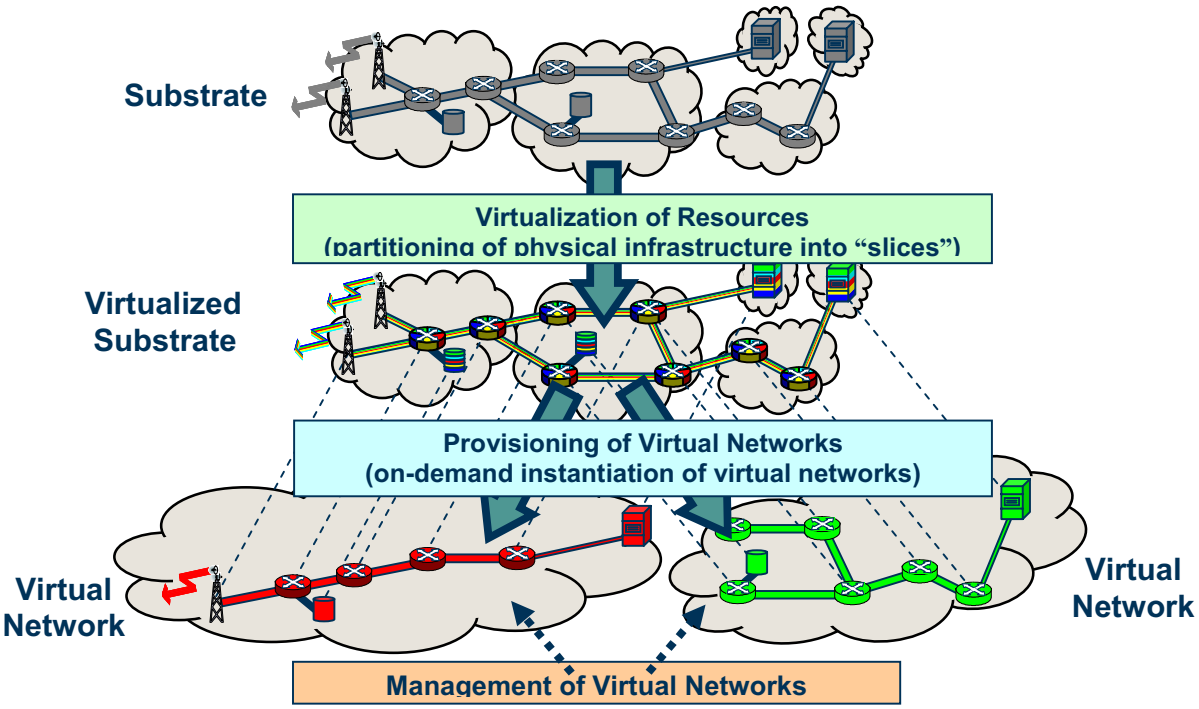
[17] http://www.ambient-networks.org/ , December 2007.

Flexible Architecture for the Future Internet based on Virtual Networks

Yasir Zaki, Carmelita Görg (Universität Bremen),
Stephan Baucke, Norbert Niebert (Ericsson)

The Internet's simple model of equally sharing bandwidth and packet processing resources is becoming a burden because some traffic profiles cannot be fully supported in an all-IP network. Resource guarantees obtained from the virtualization of physical networking resources offer the possibility of handling different profiles in parallel with the impression of mutual isolation. "Hypervisor techniques" for virtualization exist already, but the challenge is to understand the "slicing" of resources in a global networking context: The operations necessary to assemble virtual resources into end-to-end solutions must be defined and implemented in a distributed way. It should be possible, for example, for an operator to easily build, deploy and operate for customers a virtual network with performance guarantees. Isolation and assembly of independent network resources are two aspects of virtualization which is a key technique for drawing up a methodology to create a family of interoperable networks. The approach should encompass the virtualization of a broad variety of network resources, in particular including radio resources and wireless access systems

The new FP7 project 4WARD has started to explore the possibilities of the new networking paradigm. This contribution will discuss the implications with respect to the architecture, the virtualisation of resources, and the provisioning and management of Virtual Networks.



HPC-Virtualisierung und Grids

Efficient High Performance computing using Virtualisation

Stefan Boettger KIP Heidelberg
Volker Lindenstruth KIP Heidelberg
Udo Kebschull KIP Heidelberg

E-mail: boettger@kip.uni-heidelberg.de

Abstract

Usually, high performance computing aims for a maximum peak performance. However, a more reasonable measure is efficiency in terms of performance per unit of power or even computation results per node lifetime. In contrast to approaches like Blue Gene, we aim for a commodity hardware based cluster, which runs applications for both on-line and off-line processing of data gathered at the Alice-Experiment of CERN. During on-line operation times, we often encounter idle periods on certain nodes, where an off-line job could be processed until the next on-line data arrives. Since factors like depreciation, license fees and power consumption affect the total cost of ownership for a computing facility it is reasonable to run the system at nearly 100% load all the time. Running different jobs simultaneously on a native basis requires compatible installation set-ups on the computing nodes. However, many applications require different operating systems, libraries and set-ups, which normally are not compatible for installation on a single system. We therefore propose an approach based on node virtualisation in order to gain higher efficiency by keeping the cluster load as high as possible. Such a virtualisation approach also improves the usage of the cluster resources even during backup/maintenance cycles and finally, we save computing time by checkpointing and migrating of virtual servers for downtime periods. Already computed results will not be lost if a running virtualized instance is suspended and resumed later, possibly on another node. Our experiments show that we need less time and power for the same number of computed results.

1. Introduction

Scientific computing requires powerful hardware for the operating platform, which is costly in acquisition and maintenance. Therefore, for the assessment of such computing facilities, not only does peak performance have to be considered, but all costs which are subsumed under the term TCO (Total Cost of Ownership). Factors like depreciation, license fees and power consumption play an important role for computer hardware. Since license fees and depreciation are fixed on a per hardware/software item base it is desirable to exploit the available resources as much as possible, avoiding any downtimes or periods with low resource usage. Power consumption however is not fixed and dependant on the usage scheme of the hardware. For us a reasonable measure for the efficiency of a computer systems is the number of computed results per power consumption and hardware lifetime. In this paper we will propose a way of how to maximise this measure for the HLT cluster of the ALICE experiment at CERN, the ALICE Experiment aims to study the quark-gluon-plasma which presumably was present in the first nanoseconds after the big bang [12]. Both online and off-line physics data will be processed at the HLT cluster.

The attempt to use cluster hardware most efficiently, thus maximising the load of all nodes involved is not new. Job schedulers are the common way of obtaining high load on all cluster nodes. Apart from the theoretical boundaries in obtaining the best possible distribution of jobs to nodes, a satisfying average load is often not achieved in practice. For parallel applications the inter-process communication may cause the processes to underutilise the CPU, highly prioritised applications may prevent other applications from running on that node to ensure Quality of Service and applications may be compiled for different hardware/software environments and therefore not ready for usage on the same node. In dedicated clusters, like on-line clusters, applications run often, which not appropriate to run with job schedulers, since they do not comply with the requirements made by the schedulers, e.g. library usage or MPI programming style, or they use their own paradigm of distributing computations.

At the HLT cluster we want to achieve high cluster usage by utilising periods with low cluster load to

run off-line physics applications by using server virtualisation. Since HLT is an on-line cluster which uses specialised hardware to receive data from the ALICE detectors and processes these in real-time a responsible distributed application needs to utilise all resources available. However during such detector-runs there are certain intervals with little or no data arriving at the HLT. We will use these periods to run grid off-line jobs, additionally we will run off-line jobs in maintenance cycles between different runs. Since these jobs are not based on a parallel programming model such as MPI they have no latency boundaries and therefore they are perfectly appropriate for being processed during small time slots. By using server virtualisation we obtain the opportunity to run different applications with a clean separation of their environments. By checkpointing virtual machines we are able to save computing time and we will show that for our proposed setup we are able to compute more results per time and power consumption.

2. Related Work

For our purposes we need to have the ability to run an on-line application with highest priority, i.e. with all resources dedicated to that application. Approaches like Condor [2], LSF [3], NOW [4], SGE [6] try to remove, i.e. either stop or checkpoint and migrate lower prioritised jobs whenever the main application demands more resources. Stopping jobs in our environment would most certainly mean that no jobs can be finished at all, since the time-slots offered for off-line computations are smaller than the average job computation time. Checkpointing and migration in these approaches requires support for these features implemented by the job-application or the operating system, which is not true or feasible for our case. In ISF [1] and LL [5] it is proposed to have different priority classes associated to jobs to ensure the Quality of Service of the main application by modifying the kernel process scheduler. However this cannot deal with the problem of applications needing different runtime-environments (e.g. OS). Additionally uncleanly programmed low-priority jobs may interfere with other applications and affect system stability.

3. Background

The HLT cluster serves as High-Level-Trigger for the Alice-Experiment at CERN. It receives data fom all detectors involved in the experiment, processes these, calculates track-data and sends the results to the data-acquisition (DAQ).). All this is done during the runtime of the experiment. The HLT cluster currently consists of 100 nodes (AMD DualCore and QuadCore Opteron at 1.8GHz, 8GB RAM), 90 of which are being used for computations and 10 for infrastructure purposes. In 2009 the cluster will be extended up to 900 nodes with newer machines, using 8 or more cores per processor. Current interconnect technology is gigabit ethernet, Quadrics or Infiniband high performance interconnects are currently being evaluated and will be commissioned in our cluster in the near future. All nodes are equipped with specialised remote-management cards based on FPGA technology. Those CHARM (Cluster Health and Remote Management [13]) cards have their own operating system (Linux derivate) and give the remote administrator full access to the host by exporting the keyboard and VGA. All computing nodes additionally have a PCI-Card (RORC - Read-Out-Receiver-Card [12]) based on a FPGA which is connected to the detector via optical fibers and does pre-processing on the received data. The host computers run Ubuntu Linux. There a dedicated, distributed application (PubSub-Framework [7]) will do all necessary computations and initiate the send-back to the DAQ after tracks have been found. Currently the experiment is in its final test period with a starting date scheduled to may 2008. We expect to have a peak data-rate of 800MB/s per node (30GB/s in total) when the experiment is running. To cope with this amount of data all cluster ressources have to be dedicated to the responsible application to meet the time-critical requirements in calculating the track-data.

A single run of the experiment is supposed to last 24/7 for long periods (several weeks). There will be maintenance weeks between single runs. For a single run the data rate received by the HLT Cluster and thereby the load generated on cluster-side is not constant. Due to a special schedule which determines the way how and when particles are inserted into the LHC (Large Hadron Collider - the CERN particle accelerator) there will be phases at runtime where no data arrives at the HLT Cluster. The data rate arriving at the HLT is a function of the beam rate, mainly determined by the (yet variable) factors like fill length and turnaround time. A probable scenario is shown in figure 1.

Our goal is to utilise the inter-runtime as well as the inner-runtime gaps for data processing with a non time-critical application. Since HLT is a scientific physics cluster we will run the Alice Grid Framework (ALiEN) which does off-line track reconstruction. ALiEN dis-

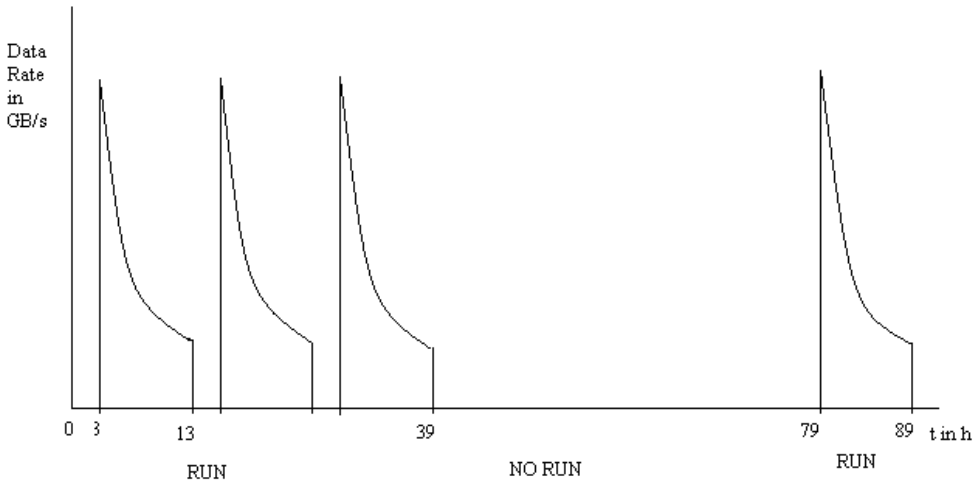


Figure 1. Data Rate Curve for Run Time

tributes its jobs to more than 30 clusters worldwide. At cluster-side a classical Job-Scheduler is required for distributing the non-parallel jobs to the nodes. Though theoretically runable on any linux OS, the current implementation is not usable on our native operating system Ubuntu. The average time needed to process a job is 8 hours [9]. So the expected gaps will not be long enough to compute a whole job.

Therefore we use server virtualisation to make best use of the available timing, os and application constraints.

4. Our Approach

The cluster was designed as a special-purpose cluster, so most of the time the PubSub-Framework will run and process detector data. For inter-run periods we will run off-line ALiEN jobs. Since ALiEN requires a Scientific Linux (SL) installation, we set up a VMware virtualised server on each of our computing nodes, equipped with SL as operating system and SGE (Sun Grid Engine) clients. This way we can also use inner-run gaps, which would not have been an option if we provided a dual-boot environment for multiple OS on the native host. To enable the use of the inner-run gaps we have to deal with the problem, that ALiEN does not provide checkpointing and the expected gaps are shorter than the average computation time of a job. We circumvent this problem by using the checkpointing / suspend feature of VMware. Since the change of the data-rate is predictable, the virtual machine (VM) is suspended and if necessary migrated to another node or simply resumed on the original host later. ALiEN does not support checkpointing by design, i.e. jobs which do not send a heartbeat signal once every 2 hours will be regarded as failed. Due to that fact the VMs which would have to be suspended for more than two hours will be moved on one hand to a special node called vmhost and resumed there for a short time, long enough to send the required heartbeat signal. Once another inner-run gap is due, the VM will be migrated back and resumed. On the other hand VMs can be reniced to use lowest processor affinity, just being able to send the heartbeat signal required to keep their jobs alive. Of course the proposed migration strategy is also applicable for maintenance actions which normally are done during the maintenance cycles between two single runs. In such a case even hardware pieces can be exchanged without loosing any computed results.

In figure 2 the comparable scenarios are shown with respect to expected power consumption and computed results (CR) for one period of the run-time and a representative part of the off-line time. The hatched area represents the number of computed results in both configurations. Our expectation is to compute the same number of results in less time with less power consumption in the virtualised scenario. To verify this we ran tests how virtualisation slows down computations, how power consumption behaves for different load scenarios and how parameters like time needed for resume/suspend of virtual machines affect overall efficiency.

Since on-line data processing does not differ we assume the number of results computed and power consumption to be the same for both configurations. Power consumption and number of results for pure

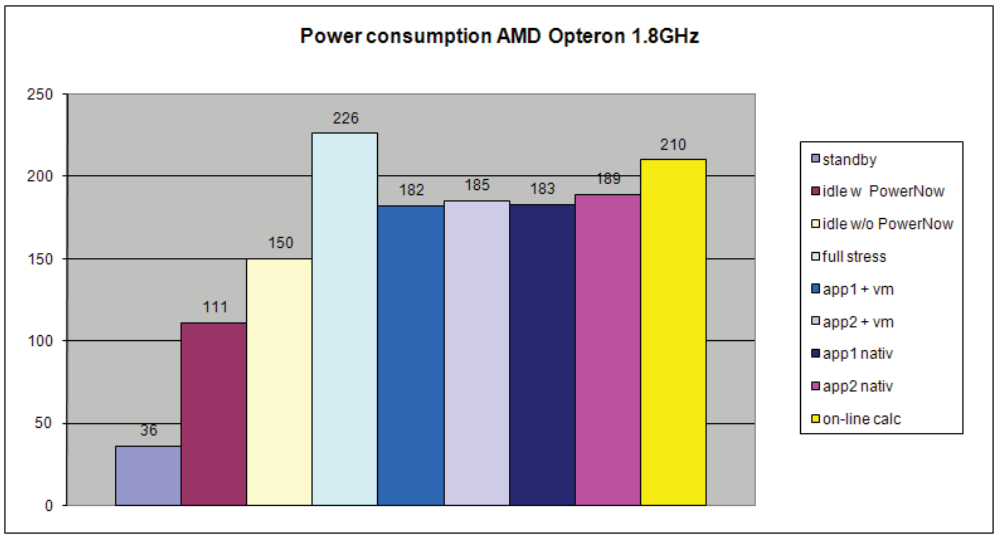


Figure 3. Power Consumption Computation Nodes

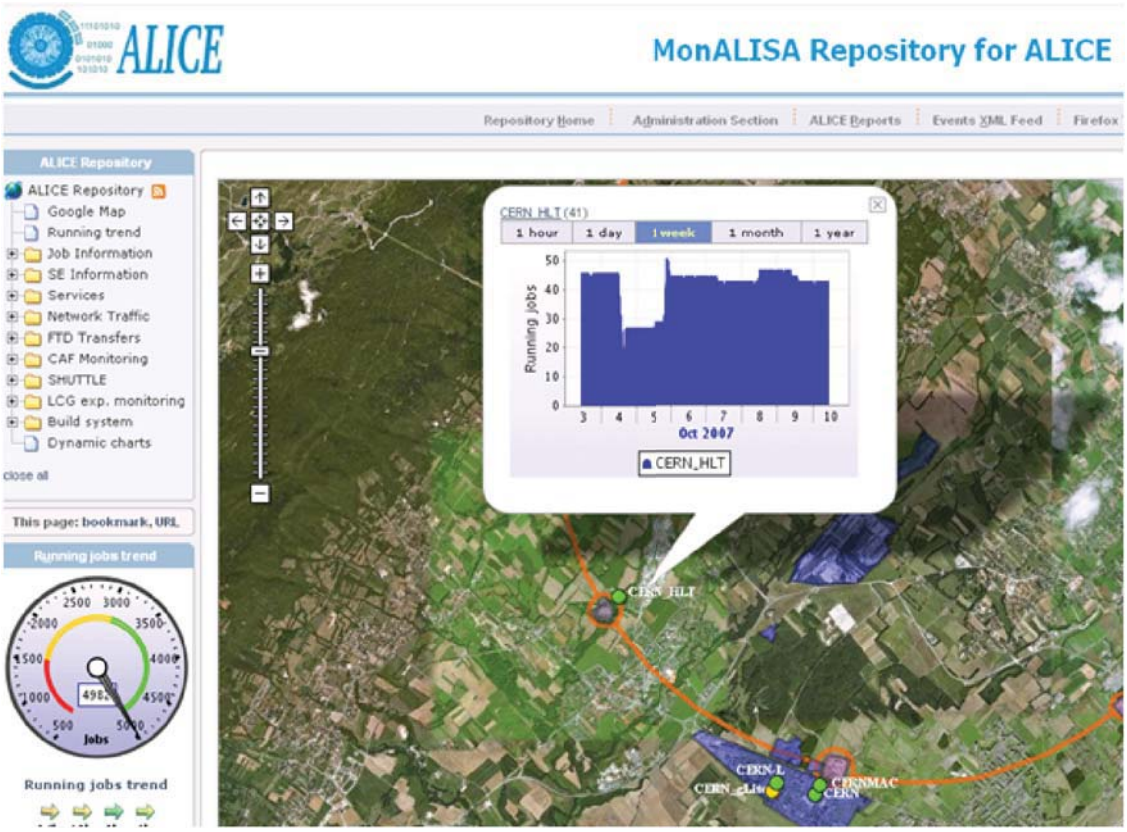


Figure 4. WebGUI for ALiEN JOB Processing

inner-run gaps and inter-run gaps depend on the load produced on a host and the computation efficiency of the nativ/virtualised environment. For testing this we ran two different kind of physics applications, one (app1) doing cluster-finding on raw event data, which causes high I/O load and the other one (app2) doing data compression which is more CPU-affine. The results are shown in table 1. It is obvious that both for

I/O and CPU-affine load the virtualisation is about 10 percent slower than calculations on a native host.

The power consumption has been measured for both applications and environments as well as for the on-line data processing. Additionally the power consumption for standby and idle mode (with/without PowerNow [10] as well as for a synthetic stress test involving both heavy I/O and CPU load have been measured. Results are shown in figure 3.

The difference between the power consumption in on-line computations and off-line computations lies in the nature of the tested applications. While the on-line application operates on all CPU cores available, the tested physics application for off-line mode only use two CPU cores. No significant differences in the power consumption for the tested applications running natively or in a vm could be found. Obviously an idle node requires less power than a fully utilised one, especially when AMD PowerNow technology is activated. Despite this observation the efficiency measure proposed by us still shows our approach to be feasible:

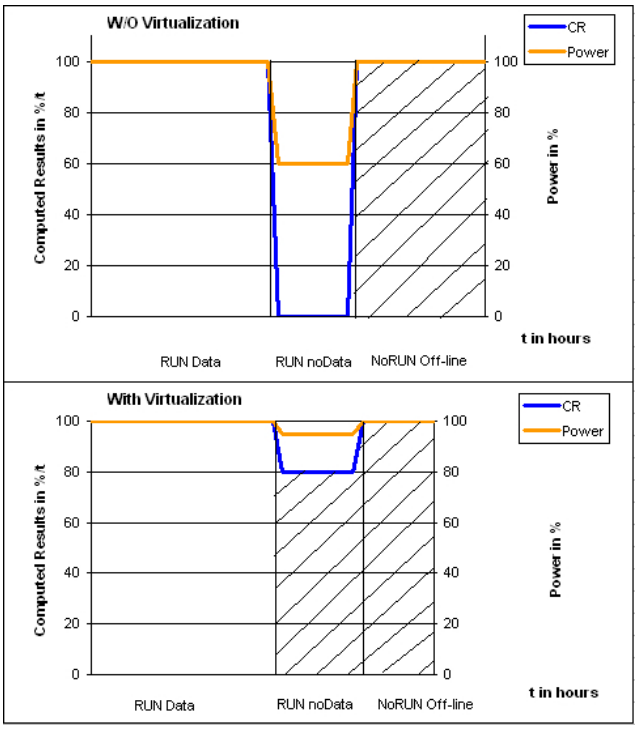


Figure 2. Power Consumption and Computed Results(CR) for Virtual vs. Nativ Scenario

Table 1. Computation Times for two Physics Applications

	Exec t in sec	
	VM	Nativ
app1 with high I/O Load	2156	1959
app2 with high CPU Load	3585	3164

efficiency measure $em = CR/(t * P)$,
with $CR = 1$ for app2 we get:

$$em_{virtual} = \frac{1CR}{3585s * 185W} = \frac{1CR}{663225Ws}$$

$$em_{nativ} = \frac{1CR}{3164s * 189W + 3585s * 111W} = \frac{1CR}{995931Ws}$$

$$\frac{em_{virtual}}{em_{nativ}} = \frac{995931}{663225} = 1.501$$

However this calculation is based on the assumption, that once the data rate drops during run-time, immediately the vm starts off-line computations and vice versa. Practically this is influenced by the time needed to resume, suspend and/or migrate a virtual machine. The following values for these processes were measured for off-line VMs having 3GB of RAM and 8GB HD:

Suspend: 114 s (vmware process finished)
+ 72 s (kjournald follow-up processing)
Resume: 113 s (till host is visible to batch scheduler)
Migrate: 308 s

The corrected efficiency calculations still show an advantage of our virtualised server configuration:

$$em_{virtual} = \frac{1CR}{(3585s + 114s + 113s + 2 * 308s) * 189W} = \frac{1CR}{836892Ws}$$

$$em_{nativ} = \frac{1CR}{(3164s * 189W + (3585s + 743s) * 111W)} = \frac{1CR}{1089504Ws}$$

$$\frac{em_{virtual}}{em_{nativ}} = \frac{1089504}{836892} = 1.301$$

Another aspect to be taken into account is the automisation of the whole process of suspending and resuming of virtual machines. In the current state of the experiment the schedule of test runs and maintenance phases is not a fixed one but subject to frequent

changes due to debugging and commissioning of all experiment detectors. So currently we suspend/resume the VMs manually, i.e. in a scripted way. For the first time when the experiment begins we will continue this and orientate at the given schedule. Currently we are working on automizing this by using the SysMES Monitoring and Management Framework [11] to monitor the data rate and other parameters indicating a appropriate moment for a vmware resume or suspend. This framework will then automatically take care of suspending, resuming and migrating the virtual machines according to predefined rules.

Picture 4 shows a web-based gui provided by ALiEN which is used to observe the state of the off-line job computations, in this particular case it shows the state of the HLT cluster.

5. Conclusion

In this paper we shown how to maximise the HLT cluster usage by using virtualisation technology. Server virtualisation not only gives us the opportunity to run incompatible applications on the same physical hosts, but also to benefit from small gaps during an on-line computation. We showed an example on a single host how the introduced efficiency measure proves our approach to be correct, i.e. we can compute more results per unit of power and time using server virtualisation.

References

[1] Gary Stiehr, Roger D. Chamberlain, Improving Cluster Performance through Intelligent Processor Sharing , Proc. of Workshop on System Management Tools for Large-Scale Parallel Systems, April 2006

[2] Litzkow, M., Livny, M., Mutka, M., Condor A Hunter of Idle Workstations, Proc. Intl Conf. on Distributed Computing Systems. June 1988, pp. 104-11

[3] Zhou S., Zheng, X., Wang, J., Delisle, P., Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, SPE, 23(12). 1993, pp. 1305-1336.

[4] Arpaci, R. H., et al., The Interaction of Parallel and Sequential Workloads on a Network of Workstations, SIGMETRICS. May 1995, pp. 267-278.

[5] Ryu, K. D. and Hollingsworth, J. K., Unobtrusiveness and Efficiency in Idle Cycle Stealing for PC Grids, in Proc. 18th Intl Parallel and Distributed Processing Symposium. April 2004.

[6] Sun grid engine (sge): A cluster resource manager. http://gridengine.sunsource.net/

[7] T. M. Steinbeck et al.,A Control Software for the ALICE High Level Trigger, , Proceedings of the Computing in High Energy Physics 2004 (CHEP04), Interlaken

[8] Dieter Rhrich, From the Big Bang to massive data flow: parallel computing in high energy physics experiments, University Bergen, Lecture Notes in Computing Science, Proceedings of the PARA2000 conference, June 2000

[9] The ALiEN GRID: http://monalisa.cern.ch

[10] AMD PowerNow Technology http://www.amd.com/us-en/assets/content_type/DownloadableAssets/Power_Now2.pdf

[11] The SysMES framework: http://wiki.kip.uni-heidelberg.de/ti/SysMES/index.php/Main_Page

[12] Torsten Alt et al., An FPGA based Preprocessor for the ALICE High-Level-Trigger, DPG Tagung , Gieen, Sektion HK (Hadronen und Kerne), 2007

[13] Ralf Panse et al., Hardwarebasiertes Computer Cluster Kontroll- und Administrationssystem, DPG Tagung , Gieen, Sektion HK (Hadronen und Kerne), 2007

Distributed Load Balancing in Heterogeneous Peer-to-Peer Networks for Web Computing Libraries

Joachim Gehweiler, Gunnar Schomaker
Heinz Nixdorf Institute and
Computer Science Department
University of Paderborn, Germany
{joge,pinsel}@upb.de

Abstract

In this work we present a novel architecture for distributed computing in a peer-to-peer network. In particular, we realize the Paderborn University BSP-based Web Computing Library (PUBWCL), which formerly used a centralized client-server architecture for scheduling and load balancing, as a pure peer-to-peer system. By using distributed heterogeneous hash tables (DHHT), our architecture features scheduling and load balancing of tightly coupled, massively parallel algorithms in the bulk-synchronous (BSP) style with a minimal number of migrations. Furthermore, our architecture is capable of heterogeneous BSP programs whereas the former version of PUBWCL could only handle homogeneous BSP programs.

Increasing Fault Tolerance by introducing Virtual Execution Environments*

Dominic Battré and **Matthias Hovestadt** and **Odej Kao**
Technical University of Berlin, Germany
{battre,maho,okao}@cs.tu-berlin.de

Axel Keller and **Kerstin Voss**
Paderborn Center for Parallel Computing
University of Paderborn, Germany
{kel,kerstinv}@upb.de

Abstract

Commercial Grids demand for contractually fixed levels of quality of service, expressed by means of Service Level Agreements (SLAs). The EC-funded project HPC4U developed transparent fault tolerance mechanisms allowing to comply with negotiated SLAs also in case of resource failures, providing checkpointing of also parallel applications and the migration over the Grid. This paper describes the concept of virtual execution environments for increasing the number of potential migration targets.

Introduction

Grid computing started under the merely technical question of how to provide access to distributed high performance compute resources. Thanks to countless projects and initiatives, funded by national and international bodies worldwide, Grid systems have significantly evolved meanwhile, making Grid technology adoptable in a large variety of usage scenarios. However, Grids are currently primarily used in the academic domain, where universities are pooling their high performance resources to Grid infrastructures and researchers are using these resources for executing applications like simulations.

Currently the Grid is on the verge of entering the commercial domain. Companies like IBM, Hewlett Packard, and Microsoft have recognized the potential of Grid Computing, investing noticeable efforts on research and the support of research communities. Already in 2003 the European Commission (EC) convened a group of experts to clarify the demands of future Grid systems and which properties and capabilities are missing in current existing Grid infrastructures. Their work resulted in the idea of the Next Generation Grid (NGG) (Priol & Snelling 2003; Jeffery *(edt.)* 2004; De Roure *(edt.)* 2006). This work clearly identified that guaranteed provision of reliability, transparency, and Quality of Service (QoS) is an important demand for successfully commercialize future Grid systems. In particular, commercial users will not use a Grid system for computing business critical jobs if it is operating on the best-effort approach only.

The EC-funded project BEinGrid (Business Experiments in Grid (BeInGrid), EU-funded Project) aims at fostering the commercial uptake of the Grid. BEinGrid encompasses numerous business experiments, where Grid technology is to be introduced to specific business domains. Some of these experiments actually reached the goal of proving the benefit of applying Grid technology for commercial customers, providing a contractually fixed level of Quality of Service. For describing such obligations and expectations within a business relationship between a service provider and a service consumer, a Service Level Agreement (SLA) is a powerful instrument (Sahai *et al.* 2002), specifying the QoS requirement profile of a job. At the Grid middleware layer many research activities already focus on integrating SLA functionality.

Modern resource management systems (RMS) are working on the best-effort approach, not giving any guarantees on job completion to the user. Since these RMS are offering their resources to Grid systems, Grid middleware has only limited means in fulfilling all terms of negotiated SLAs. For closing this gap between the requirements of SLA-enabled Grid middleware and the capabilities of RMS, HPC4U (Highly Predictable Cluster for Internet-Grids (HPC4U)) started working on an SLA-aware RMS, utilizing the mechanisms of process-, storage- and network-subsystems for realizing application-transparent fault tolerance. The RMS OpenCCS has been selected as a central component of the HPC4U project, since the planning based nature of OpenCCS seemed to be well-suited for realizing SLA-awareness. Within the project all features required for SLA-awareness and SLA-compliance have been developed, e. g. an SLA-aware scheduler, mechanisms for transparent checkpointing of parallel applications, or the negotiation of new SLAs.

For increasing the level of fault tolerance, the HPC4U system has been enabled to migrate checkpointed jobs between cluster systems, i. e., between cluster systems within the same administrative domain or even to arbitrary cluster resources within the Grid. Prior to such a migration process, the remote cluster system has to agree on all terms of the job that is to be migrated. This ensures that the job will be com-

*This work has been partially supported by the EU within the 6th Framework Programme under contract IST-031772 "Advanced Risk Assessment and Management for Trustable Grids" (AssessGrid) and IST-511531 "Highly Predictable Cluster for Internet-Grids" (HPC4U).

pleted as agreed with the service customer. Since all fault tolerance mechanisms are transparent, the customer will not notice that the job has been completed on a remote cluster system.

However, it is not possible to use arbitrary cluster resources for resuming the checkpointed job. Since the job has been started and checkpointed in the context of a specific execution environment, the remote resource has to be compatible to this execution environment. These compatibility demands regard high level requirements like processor architecture and operating system, but also low level aspects like availability and version of libraries. In HPC4U all these demands have been expressed within a compatibility profile, which is part of the SLA negotiation process with the remote cluster resource. Even if this compatibility profile significantly increases the chance of successfully resuming checkpointed jobs on remote cluster systems, it also significantly reduces the number of potential migration targets. Even large heterogeneous Grid systems only have a very small number of compatible resources which can be used as potential migration targets. The number of available migration targets is even smaller, since the remote cluster system has to agree on all other terms of the SLA, e. g. the compliance with deadlines.

Obviously increasing the number of compatible resources would also increase the number of potential migration targets, thus increasing the level of fault tolerance. For achieving this goal, we introduce the instrument of virtual execution environments, which are then established at the compute resources, ensuring the compatibility of the compute resource with the migrated job.

In this paper, we will first highlight the architecture of the HPC4U cluster system and the demands on compatibility. The main part will then describe the instrument of virtual environments. The paper ends with an overview about related work and a short conclusion.

HPC4U Architecture

The HPC4U cluster middleware consists of multiple elements, i. e., the SLA-aware resource management system and the main building blocks for ensuring a high level of fault tolerance: process checkpointing, storage snapshot and virtualization, and network failover. In an exceptional situation, e. g. the outage of hardware resources, the HPC4U system uses its FT (Fault Tolerance) mechanisms to assure the completion of a job. This means that the Metacluster software enables checkpoint/restart (and migration) of a running process, so that jobs can be restarted from the last checkpoint on a spare resource. But only considering the checkpoint process could cause inconsistencies at restart because the checkpoint's data and job's data can be at a different stage as a running job continues to write data on files after the checkpoint. Therefore, the system has to maintain consistency between checkpoints' data and job's data. This process has also to be supported by the network subsystem, e. g. regarding in-transit network packets.

The results of HPC4U are a mix of open source and proprietary software embedded in three outcomes (cf. Figure 1) (Heine, Hovestadt, & Kao 2004). The SLA-aware and Grid-enabled Resource Management System includes SLA negotiation, multi-site SLA-aware scheduling, security and interfaces for storage, checkpointing, and networking support. It is available for multiple platforms and distributed as open source. The second HPC4U outcome is a vertically integrated commercial product with proprietary Linux-specific developments for storage, networking, and checkpointing. This outcome demonstrates the entire, ready-to-use HPC4U functionality (job checkpointing, migration, and restart) for Grids based on Linux architectures. It is obvious that providing an agreed level of Quality of Service and Fault Tolerance requires broad interaction between all components of the HPC4U system. The third outcome also depicts a vertically integrated system, but consisting of non-commercial components only. Compared to the commercial system this system has significant functionality drawbacks but can be easily evaluated without the need of obtaining any licenses.

Without loss of generality we assume that a user from somewhere in the Grid wants to compute a job and connects to an HPC4U system for negotiating on a Service Level Agreement. Usually, a user would not connect directly to an HPC4U system, but uses his local Grid middleware interface for finding suitable resources for his request. Matchmaking mechanisms on the level of Grid middleware compare requirements with published information about available resources. Hence, Grid middleware mechanisms offer intermediary services. However, from the point of view of an HPC4U system, it makes no difference if a user or some Grid middleware element starts a service negotiation request.

The cluster middleware system consists of three independent layers:

- At the upper layer, the system provides an interface, which can be used by Grid middleware systems to negotiate on Service Level Agreements. This interface applies to standard protocols used in Grid middleware ensuring interoperability with other projects.
- At the middle layer, an SLA-aware resource management system using the upper layer interface, negotiating with customers on SLAs. It also assures the compliance with these agreed SLAs at runtime. This does not only imply the monitoring of internal resources, but also the utilization of appropriate mechanisms to realize fault tolerance in case of resource outages.
- At the lower layer are the subsystems of HPC4U. Offering specific APIs, each of these subsystems provides special mechanisms for fault tolerance on process-, network- or storage-level. Since all interfaces within the HPC4U system are published, each component can be replaced with arbitrary third-party products, as long as these products provide compliant interfaces.

Compatibility Profile

The SLA-aware resource management system uses process checkpoints for various purposes. Beside the compensation of local resource outages by intra-cluster migration, a checkpoint may also be transferred to remote systems. Kernel-level checkpointing systems allow the checkpointing of arbitrary applications without the need of prior relinking or

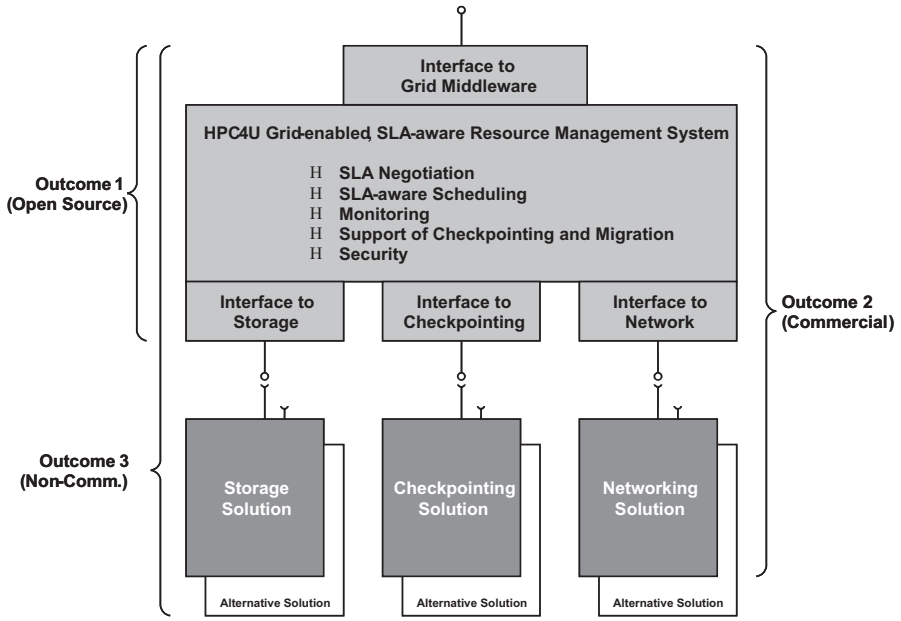


Figure 1: Outcomes of the HPC4U project

recompiling. When focusing on commercial users, who execute proprietary applications, this is necessary as relinking or recompiling is not be possible in most cases. This flexibility and transparency on the other hand has the drawback of a high degree of system dependence.

In contrast to application level checkpointing, a kernel-level checkpointed process can not be restarted on arbitrary target systems. Beside high level characteristics like operating system or processor type, the target machine even has to be compatible in regard of versions of installed libraries and tools. If restarting a checkpointed job on an incompatible resource, the job would directly crash at best. In the worst case, the application would resume its computation, but return incorrect results. In this case the RMS would assume that the application restarted successfully, returning incorrect results back to the customer.

An obvious way to face this situation and ensure a successful restart on the target machine is to request identical machines. At this, the hardware of a suitable target machine must be identical to the source machine. The same holds for the software installation. Both machines have to have identical operating systems with identical upgrade levels (e. g. RedHat AS4, Upgrade 4).

Even if this strict demand solves the problem of compatibility very efficiently, it reduces the number of eligible target systems in a migration process close to zero. If looking to resource information catalogues in the Grid, a broad variety of different systems becomes apparent. Even if some of these systems would be able to resume the checkpointed application, this strict demand on equality would disqualify them.

For enhancing the number of potential migration targets while ensuring their compatibility, the compatibility profile is introduced. This profile is an instrument for describing the application's requirements on the target machine, so that the restart can be successful.

The compatibility profile holds information regarding the general demands on the target resource. The most fundamental requirement on the target machine is regarding its internal architecture and system properties. These demands are not specific to the used checkpointing system, but arise from the execution environment. Furthermore, the operating system installed on a compute node forms the fundament for the application execution, e. g. the execution within a Linux operating system.

All Kernel-level checkpointing solutions have their general functionality in common. By intercepting specific system calls they allow to generate a process image of a running application. Despite the fact that these solutions differ in their particular functionality profiles, it is not possible to exchange checkpoint datasets between them. Therefore it is necessary to have the same checkpointing solution available on the target machine that was used to generate the checkpoint. Hence, the checkpointing system is also part of the compatibility profile.

Beside operating system, checkpointing system and processor architecture, a broad variety of other system properties is essential for a successful restart of the application, e. g. sufficient amount of main memory as well as storage capacity. Also hardware demands like the availability of a specific network interconnect, or software demands like special purpose applications, libraries, or licenses are part of the customer agreed SLA. The SLA may also demand for the availability of a specific filesystem type. In this case, also the filesystem must be available on the target machine.

The concept of libraries is known in almost all operating systems. Instead of demanding each application developer to write the same core functions again and again, these func-

tions are provided by means of libraries. The operating system itself is offering system services over system libraries. By linking his application against these libraries, this the programmer is able to use these functionalities easily.

Static libraries are linked to the application and part of the resulting binary. This way, the user does not have to ensure the availability on the system where he plans to execute the binary. However, static linking results in significant waste of space, both storage and memory. Furthermore this type of library complicates system maintenance. On updating a given library (e. g. due to a security problem or programming bug), all applications using this library have to be relinked.

Shared libraries in contrast are only loaded once into the system memory. On application start, the availability of the library is checked by a loader service. This loader verifies the version of the library, sets entry addresses, and maps the memory of the library to the virtual memory segment of the application.

Dynamic loading further improves the concept of a shared library. It allows the application to dynamically load and unload a library at runtime. Beside performance increase at start time, this method also has the advantage that applications can start even if specific libraries are not available on that system and not crucial for program execution.

From the checkpoint compatibility point of view, dynamic loading is a serious issue, because libraries are not necessarily placed at the same position in memory at each restart. If a checkpoint is resumed in an environment where these libraries are loaded to different memory addresses, the application would access the wrong memory segments at runtime.

Currently available checkpointing solutions are solving this problem by saving the address of these libraries to the checkpoint dataset file. If restarting the application on a remote system, the system checks the addresses of these libraries. If necessary, it then reloads the library and maps the addresses for the restarted application. However, this method requires the library to be installed at the same position (i. e., directory path) and in the same version. Due to this reason it is important to add information about the required libraries to the compatibility profile.

In the Linux operating system libraries are stored having their version in their filename. The library can be found under its major version number due to a link from the real library name to the virtual library name, which only holds the major version in its name (e. g.`libcap.so.1 -> libcap.so.1.10`, or `libnetsnmp.so.5 -> libnetsnmp.so.5.1.2`).

Changes in the patch version usually do not refer to changes in the functions, so that programs running with version `5.1.2` should also restart with `5.1.1`. Minor version changes signal a change in functions, which is backwards compatible to older versions, so that `1.10` should not be restarted with `1.9`.

However, it has to be distinguished between loaded and unloaded libraries at this point. If a library has been loaded, the loader service of the operating system mapped all addresses according to the particular library version. Since this address mapping information is part of the checkpoint dataset, the job would also use the same information at restart.

Even minimal differences in the code of a library has effect on the memory size of that function. The result is that address mapping tables are different between two patch versions. If the job restarts with an address mapping table, that does not match with the installed library, this would cause an incorrect behavior at runtime.

Therefore the checkpointing profile has to distinguish between loaded and unloaded libraries.

- For unloaded libraries it is sufficient to query for a compatible library version.
- For loaded libraries it is mandatory that the identical version is available on the migration target system.

The resource management system can retrieve the library related information about a running job by analyzing the application and checking the list of loaded libraries at checkpoint time. According to this list of libraries the RMS is then able to check the version numbers of the libraries, adding either the full version or solely the major version number to the compatibility profile.

Virtual Execution Environments

The previous section underlined the difficulties of retrieving compatible resources for migration within Grid systems. Only if the target resource matches the compatibility profile, the checkpointed job may be migrated.

It is common practise that Grid resources are operated under a decentralized autonomy of local resource providers. Hence, local administrators decide on the operating system to be installed on compute resources. Moreover, is is the responsibility of local resource administrators to install updates, e. g. applying available patches to the local compute node environment. This local autonomy is the root cause of the difficulty of finding compatible resources. Since each difference in the operating system patchlevel impacts the compatibility for the migration process, it would be beneficial to have commonly accepted and available execution environments, guaranteeing the compatibility regarding paths and libraries.

This goal can be achieved by using virtualization technology on the compute resources. Instead of executing applications directly on a compute node, a virtual system is started first. Within this virtual system arbitrary operating systems can be started. The application then is started within this virtual system environment. This way, it is possible to start the application within a well-defined system environment.

The central component of this new infrastructure is a central system image repository, holding system images of different operating systems, e. g. commonly used Linux distributions in different versions and different patchlevels (cf. figure 2). Each of these images has a unique ID. The contents of this image repository can be published within the Grid using well established mechanisms like resource information catalogues. Since these information services are public, not only a single provider is able to access the contents, but all Grid stakeholders.

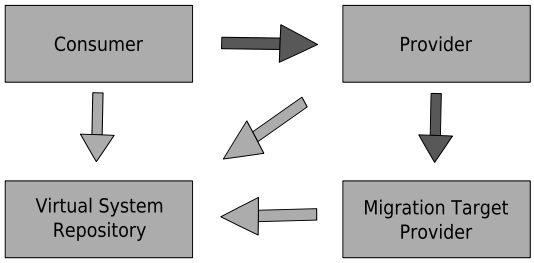


Figure 2: Virtual System Repository

Already at SLA negotiation time, the service customer now has the opportunity to specify a system image to be established at execution time, instead of the current practise of not knowing if the job will be executed on an up-to-date Debian system or an outdated SuSE. Hence, the customer can test his job in the specified environment by establishing the image from the system image repository locally, being sure that the job will succeed and return the expected results.

Providers will typically only support a subset of images from the image repository, e. g. images of distributions that are known by the system administrators, or that have proven to run stable on the compute resources. Providers are free in the selection of supported system images, which are then published in the Grid resource information catalogue.

At level of Grid middleware the customer requested virtual system ID has to be matched against the provider supported characteristics of their resources, like it is already common practise with all other system parameters like number of nodes or amount of main memory. Hence, currently existing matching mechanisms, e. g. used by Grid broker systems, can be used for also matching the virtual system IDs.

If a provider agrees on an SLA specifying a virtual system image ID, it has to establish the specified system image at runtime on the compute node and then execute the user job within the specified environment. If the customer did not specify any image ID, the provider may choose a default virtual system to be executed on the compute node. In this case, the customer does not have any knowledge about the system that is used for executing his job, but this only corresponds to the current situation.

At runtime, the virtual system is then established on the compute node, so that all job checkpoints can be executed in a well defined and well known environment. Hence, all dependencies of the checkpointed job on the system environment are known, since the system has been executed in the specified virtual environment.

This is particularly beneficial for the migration process, since it is no longer mandatory to generate a compatibility profile, as explained in the previous section. Instead of querying for libraries or library versions, the resource provider is able to query for resources that can execute the particular virtual system ID. Hence, instead of describing requirements on a compatible environment, now the compatible environment can be requested directly.

This virtual system ID is part of the SLA negotiation process of the source and the target resource provider, which is providing the migration target resources. If the target resource provider agrees on the SLA, it agrees on establishing the specified virtual system on the compute node at runtime, where the checkpointed job is to be resumed. This mechanism also applies to all further migration operations of this job, e. g. if the new resource provider again has to query the Grid for backup resources due to resource failures.

Related Work

The worldwide research in Grid computing resulted in numerous different Grid packages. Beside many commodity Grid systems, general purpose toolkits exist such as Unicore (UNICORE Forum e.V.) or Globus (Globus Alliance: Globus Toolkit). Although Globus represents the de-facto standard for Grid toolkits, all these systems have proprietary designs and interfaces. To ensure future interoperability of Grid systems as well as the opportunity to customize installations, the OGSA (Open Grid Services Architecture) working group within the OGF aims to develop the architecture for an open Grid infrastructure (GGF Open Grid Services Architecture Working Group (OGSA WG) 2003).

In (Jeffery *(edt.)* 2004), important requirements for the Next Generation Grid (NGG) were described. Among those needs, one of the major goals is to support resource-sharing in virtual organizations all over the world. Thus attracting commercial users to use the Grid, to develop Grid enabled applications, and to offer their resources in the Grid. Mandatory prerequisites are flexibility, transparency, reliability, and the application of SLAs to guarantee a negotiated QoS level.

An architecture that supports the co-allocation of multiple resource types, such as processors and network bandwidth, was presented in (Foster *et al.* 1999). The Globus Architecture for Reservation and Allocation (GARA) provides ”wrapper” functions to enhance a local RMS not capable of supporting advance reservations with this functionality. This is an important step towards an integrated QoS aware resource management. In our paper, this approach is enhanced by SLA and monitoring facilities. These enhancements are needed in order to guarantee the compliance with all accepted SLAs. This means, it has to be ensured that the system works as expected at any time, not only at the time a reservation is made. The GARA component of Globus currently does neither support the definition of SLAs or malleable reservations, nor does it support resilience mechanisms to handle resource outages or failures.

The requirements and procedures of a protocol for negotiating SLAs were described in SNAP (Czajkowski *et al.* 2002). However, the important issue of how to map, implement, and assure those SLAs during the whole lifetime of a request on the RMS layer remains to be solved. This issue is also addressed by the architecture presented in this paper.

The Grid community has identified the need for a standard for SLA description and negotiation. This led to the development of WS-Agreement/-Negotiation (Andrieux *et al.* 2004).

The usage of virtualization technology at provider level within resource management systems has been described in (Fallenbeck *et al.* 2006). Here the Xen virtual machine monitor is used for increasing system utilization and response time of a cluster system operated with the Sun Grid Engine resource management system. Long running sequential jobs are executed in a virtualized environment and suspended for executing short running parallel jobs. This work underlined the general applicability of virtualization technology on compute nodes, but did neither address fault tolerance nor the execution of parallel applications within virtual environments.

Conclusion

SLA-awareness is a mandatory prerequisite if the commercial user should be attracted to use Grid environments. Since SLA-awareness does not only focus on the negotiation of new SLA but also on their fulfillment, mechanisms for providing SLA-compliant service also in the case of resource failures are required. The EC-funded project HCP4U aims at providing an application-transparent and software-only solution of such an SLA-aware RMS, demanding for reliability and fault tolerance. The HPC4U system already allows the Grid user to negotiate on new SLAs, which will be realized by means like process-, network,- and storage-checkpointing.

Migrating checkpoint datasets over the Grid to remote cluster resources particularly implies requirements on the compatibility of source and target resource. Only if the target resource matches the source resource to a large extend, the checkpoint will be able to resume successfully. For describing all these requirements, the HPC4U introduced the compatibility profile. At migration time, the system has to query for target resources matching the terms of this profile.

Due to these fine grained requirements the number of matching resources is fairly small even in large heterogeneous Grid systems. Introducing virtual execution environments on the compute nodes allows to address the question of resource compatibility in a novel way. Instead of describing the requirements of a compatible environment, the compatible environment itself is established on a compute resource at resume time of a checkpointed job.

The mechanisms presented in this paper preserve the local autonomy of resource administrators, who still can decide which virtual system should be supported. Moreover already available resource querying mechanisms within the Grid can be used for finding resource providers supporting a specific virtual execution environment. The implementation of virtual execution environments is subject of current work, using the resource management system OpenCCS and the Xen virtual machine monitor.

References

Andrieux, A.; Czajkowski, K.; Dan, A.; Keahey, K.; Ludwig, H.; Nakata, T.; Pruyne, J.; Rofrano, J.; Tuecke, S.; and Xu, M. 2004. Web Services Agreement Specification (WS-Agreement). `http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf`.

Business Experiments in Grid (BEinGrid), EU-funded Project. `http://www.beingrid.eu`.

Czajkowski, K.; Foster, I.; Kesselman, C.; Sander, V.; and S.Tuecke. 2002. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In D.G. Feitelson, L. Rudolph, U. S. E., ed., *Job Scheduling Strategies for Parallel Processing, 8th InternationalWorkshop, Edinburgh,*.

De Roure *(edt.)*, D. 2006. Future for European Grids: GRIDs and Service Oriented Knowledge Utilities. Technical report, Expert Group Report for the European Commission, Brussel.

Fallenbeck, N.; Picht, H.-J.; Smith, M.; and Freisleben, B. 2006. Xen and the Art of Cluster Scheduling. In *Second International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*.

Foster, I.; Kesselman, C.; Lee, C.; Lindell, B.; Nahrstedt, K.; and Roy, A. 1999. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *7th International Workshop on Quality of Service (IWQoS), London, UK*.

GGF Open Grid Services Architecture Working Group (OGSA WG). 2003. Open Grid Services Architecture: A Roadmap.

Globus Alliance: Globus Toolkit. `http://www.globus.org`.

Heine, F.; Hovestadt, M.; and Kao, O. 2004. HPC4U: Providing Highly Predictable and SLA-aware Clusters for the Next Generation Grid. In *4th Cracow Grid Workshop, Cracow, Poland*.

Highly Predictable Cluster for Internet-Grids (HPC4U), EU-funded project IST-511531. `http://www.hpc4u.org`.

Jeffery *(edt.)*, K. 2004. Next Generation Grids 2: Requirements and Options for European Grids Research 2005-2010 and Beyond. `ftp://ftp.cordis.lu/pub/ist/docs/ngg2_eg_final.pdf`.

Priol, T., and Snelling, D. 2003. Next Generation Grids: European Grids Research 2005-2010. `ftp://ftp.cordis.lu/pub/ist/docs/ngg_eg_final.pdf`.

Sahai, A.; Graupner, S.; Machiraju, V.; and van Moorsel, A. 2002. Specifying and Monitoring Guarantees in Commercial Grids through SLA. Technical Report HPL-2002-324, Internet Systems and Storage Laboratory, HP Laboratories Palo Alto.

UNICORE Forum e.V. `http://www.unicore.org`.

From CIM to GLUE: Translate Resource Information of Virtual Machines to Computational Grids

Lizhe Wang Marcel Kunze Jie Tao

Institute for Scientific Computing (IWR), Research Center Karlsruhe (FZK)
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
{Lizhe.Wang, Marcel.Kunze, Jie Tao}@iwr.fzk.de

Abstract

Distributed virtual machines can help to build scalable, manageable and efficient Grid infrastructures. The work proposed in this paper focuses on information management for virtual machine based Grid systems. Resource information from popular virtual machine products, e.g., Xen and VMware, is organized in CIM schema. The Grid computing community, on the other hand, in general employs another information schema, such as GLUE. To manage virtual machine resources for Grid computing, efficient information transfer of virtual machine resource to Grid high level services is required. In the paper an information service is built to retrieve resource information of virtual machines, and translate CIM based information to Grid information defined in GLUE schema. The resource information is finally transferred to high level Grid services, for example Web interface access for users. We argue that the implementation is the first attempt of organizing virtual machine information for Grid computing. The work is implemented in a test bed and shown with example usage.

1. Introduction

Grid computing technology [12] offers promising solutions for parallel and distributed computing. It can provide reliable, collaborative and secure access to remote computational resources as well as distributed data and scientific instruments.

A virtual machine is a computing platform that creates a virtualized layer between the computing hardware and the application. This paper is devoted to discuss a Grid workflow system on distributed virtual machines. There are advantages of using virtual machines, like on demand creation and customization, performance isolation, legacy software support and ease of management.

Virtual machine based Grid systems are characterized by some special features, which bring research challenges for deploying, monitoring and operating the system:

- Site autonomy
  In the virtual machine based Grid system, the hosting resources, which run a Virtual Machine Monitor (VMM) and support multiple virtual machines, are commonly owned and controlled by different institutes or organizations at different sites. Users may expect to meet different resource management policies during the creation and manipulation of virtual machines.

- Hierarchy
  A virtual machine based Grid system is hierarchical in nature. It contains several levels, virtual machine level, hosting resource level and the user access point, i.e., Grid portal.

- Heterogeneity
  A virtual machine based Grid system includes heterogeneous hosting resources, virtual machine technologies (e.g., Xen, VMWare) as well as programming interfaces.

- Large scale distribution
  Computer centers and data centers frequently employ virtual machines and build Grid infrastructures across geographically distributed sites.

In a virtual machine based Grid system, some specific requirements demand attention for the information service:

- Efficient delivery of resource information from virtual machines to clients in the hierarchical Grid environment;

- Information services should be scalable and robust with regard to dynamic startup/shutdown of virtual machines;

- The information collector which runs inside in the virtual machine should be lightweight, portable and manageable.

- Information from popular VMM such as Xen and VMware which is defined by the CIM schema should be translated to higher level Grid services and user access.

This work implements an information collector, a translator and an information provider for VMware virtual machines and builds a Grid information service which can retrieve resource information from the information provider. The work is to our knowledge the first prototype of an information service for a virtual machine based Grid system, which can translate CIM based virtual machine information into Grid information defined in GLUE schema.

The paper is organized as follows: related work is investigated in Section 2; Section 3 give an overview on the design and implementation of the information service. Section 4, Section 5 and Section 6 detail the implementation of components in the information service: the information collector, the translator, and the information provider. In Section 7 test results are presented and discussed. Section 8 concludes the paper and points out future work.

2. Related work

Since several years the Grid computing research community shows interest for virtual machines and virtual environments. The typical Virtual Machine Monitor (VMM) or hypervisor setup includes Xen VMM [2], VMware server/ESX server [27], and User Mode Linux [16]. In general, users can benefit from the virtualization techniques in the following aspects:

- On demand creation and customization
  Users can create a customized virtual machine, which can provide customized resource allocation for users, e.g., OS, memory, storage, etc.

- Performance isolation
  Virtual machines can guarantee the performance for users and applications. Users of virtual machine could expect a dedicated computing environment, which is hard to find in multiple-user computing servers.

- Legacy software support
  Customized virtual machines which are compatible with legacy binary applications can be created. Users from specific engine domains can find it very desirable and promising since some legacy libraries could be supported.

- Easy management
  Users in general should only access computing servers with restricted user privilege. It is thus difficult to process the work such as compilation, installation, configuration of desirable computing environment for users. Virtual machines on the contrary, could offer users with “root” access of the allocated virtual machine. Therefore application domains could manage their environments in their own interest.

The Globus alliance recently implemented the concept of virtual workspace [15] which allows a Grid client to define an environment in terms of its requirements, manage it, and then deploy the environment on the Grid. The implementation is based on Globus Toolkit 4 (GT4) and it only supports Xen VMM. Some other research work also focuses on deploying computing systems or test beds with virtual machines, for example, virtualization of batch queueing system [3], GridBuilder [4], using virtual machine as Grid gateway [5], multi-site MPI platform with Xen virtual machine [24], migration of virtual machines in MAN/WAN [26].

Other researchers try to build virtualized middleware for clusters and distributed systems. Xen Grid Engine [9] follows an approach to create dynamic virtual cluster partitions using para-virtualization techniques. The work presented in [19] builds virtual clusters and virtualized distirbuted infrastructures. The In-VIGO [1] project aims to build virtulization middleware for computational Grids. In-VIGO provides a distributed environment where multiple application instances can coexist in virtual or physical resources such that clients are unaware of the complexity inherent to Grid computing.

Various advances have been made in field of virtual network. Violin [19]employs user-level communication indirection between the virtual machines and the underlying infrastructure. The In-VIGO system implements Wide-Area Overlays of Virtual Workstations (WOWs) by creating virtual IP networks on top of P2P overlays. While the Violin and the In-VIGO implementations are based on user-level overlay networks, VNET [23] is realized in both user-level and kernel level: host kernel-level devices are created to tunnel network traffic.

Another important topic is the performance analysis of virtual machines or virtual environments. The Xen group has published a performance evaluation and comparison between several popular VMMs concerning the performance overhead in different scenarios [17]. Other research efforts refer to virtual machine based systems, i.e., performance of para- and paene- virtualized systems [20], performance enhancement of SMP clusters with virtual machines [22].

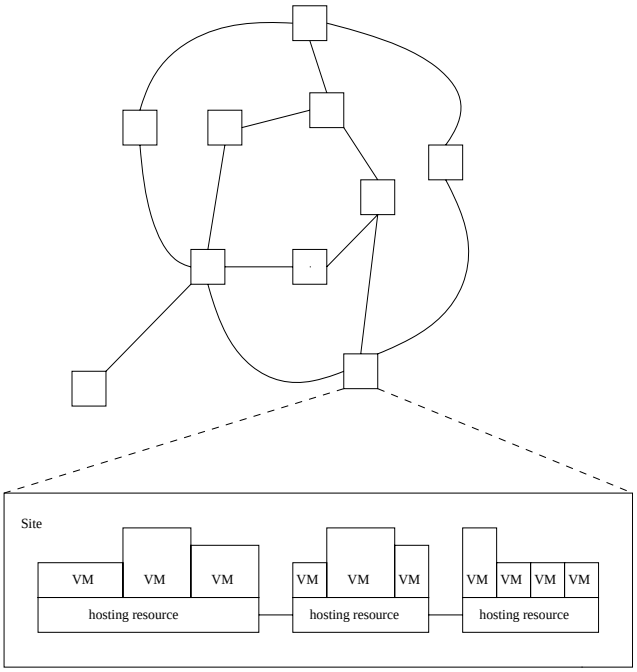


Figure 1. Hierarchy of the virtual machine based Grid system

3. System Overview

We propose a system model to describe a distributed, hierarchical, heterogeneous virtual machine based Grid system. The system architecture has been described hierarchically (see also Fig. 1):

- Grid level
  The target Grid system contains multiple geographically distributed sites, which could be computer centers, data centers, universities, and research institutes. On the Grid level, each site is represented and accessed via an access point. In other words, users can submit jobs to some computing centers and get their resource information via the access point.

- Site level
  Each site provides a number of physical resources, for instance, cluster, PVP, and MPP. Resources in each site are connected with LAN and can support multiple virtual machines.

- Virtual machine level
  Grid users can demand virtual machines on hosting resources, submit jobs to virtual machines, monitor their jobs and the guest systems.

The information service consists of an information collector in virtual machine, the client of information collector, and information translator and an information provider in the access point, and the aggregated Globus index service. The information collector which runs inside a virtual machine is used to retrieve resource information for the information provider. Information collector clients get results from information collectors, the information translators change CIM information to GLUE information and information providers for Globus MDS 4 (Monitoring and Discovery System) organize the resource information from information collectors in predefined XML schema, which are aggregated into the Globus index service. The WebMDS can be configured as a graphical user interface based on the Globus index service (see also Fig. 2).

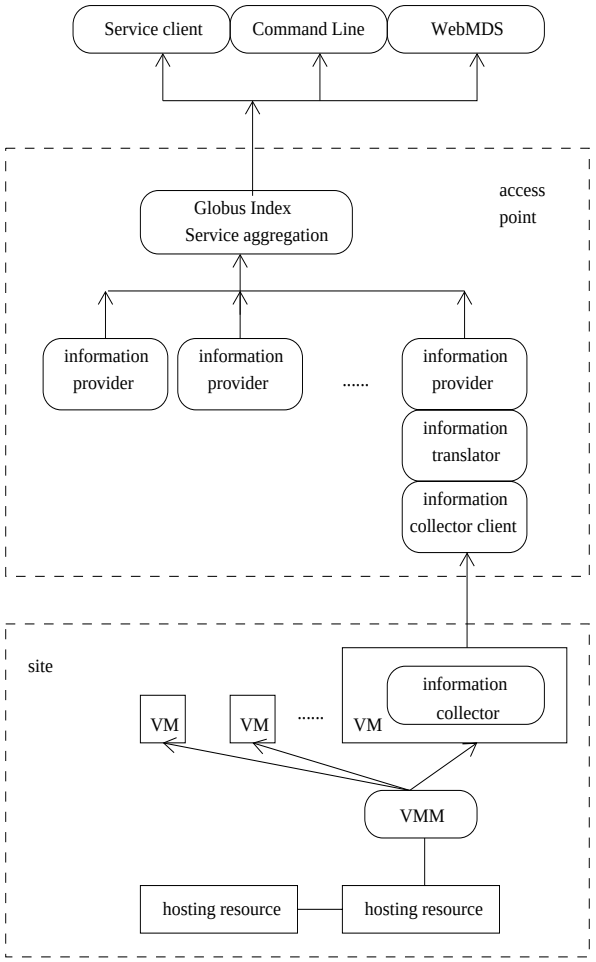


Figure 2. Overview of the information service

4. Information Collector

The information collector is a light weight software, which resides inside a virtual machine and collects resource information. In the current implementation, we programm on VMWare ESX server APIs and build information collector on VMware virtual machines.

VMware ESX server is a commercial virtualization product of VMware Inc. Common Information Model (CIM) [6] is defined as an international standard by the Distributed Management Task Force (DMTF) [11]. The VMware ESX server together with CIM SDK provides a CIM-compliant object model for virtual machines and their related storage devices. Fig. 3 shows a typical configuration environment of VMware ESX server. The virtual machine contains a virtual disk that resides as a virtual disk file on a storage area network.

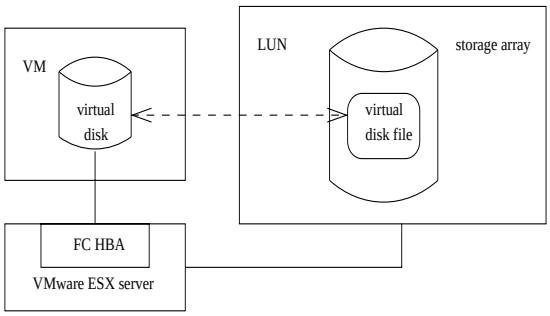


Figure 3. Sample environment of VMware ESX server

The SMI-S (Storage Management Initiative Specification) schema for the sample VMware ESX server environment is shown using UML in Fig. 4. *ESXComputerSystem* is the kernel object of the system. It associates *VM*, *VirtualDisk* and *FC HBA&LUN*[1] with *HostedDependency*, *HostedStoragePool* and *SystemDevice* relationships respectively. *VM* is associated with *VirtualDisk* with *ArchiConnection* relationship. The latter is associated with *FC HBA&LUN* in *ConcreteComponent* relationship.

The pegasus CIMOM (CIM Object Manager) [7] is deployed on the VMware ESX server. The information provider at the access point works as CIM client and communicates with pegasus CIMOM to retrieve information from virtual machines and their associated storage. The information provider complies with SMI-S profile [21] and transports CIM XML over HTTP/HTTPS to pegasus CIMOM (see also Fig. 5).

We implement the client side codes which communicate with pegasus CIMOM server on the basis of VMware ESX server APIs. The communication between the client and pegasus CIMOM server is based on HTTP/HTTPs or TCP/IP protocols. The client side codes, together with information translator and information provider, reside in the access point of the site.

[1] Fibre Channel Host Bus Adaptor & Logical Unit Number

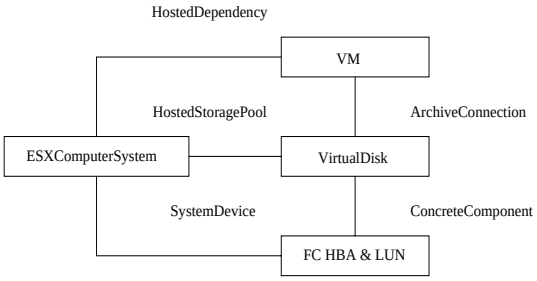


Figure 4. CIM schema for VMware ESX server

Figure 5. CIMON for VMware ESX server

5 Information Translator

5.1 CIM schema for virtual machine information

The CIM (Common Information Model) schema [6], which is a standard created by DMTF (Distributed Management Task Force) provides a common definition of management information for systems, networks, applications and services. CIM is a conceptual information model for describing computing and business entities in enterprize environments. The fundamental goals of CIM are common definitions that enable vendors to exchange semantically rich management information between wide varieties of systems.

The VMware CIM SDK provides a CIM interface for developers building management applications. With the VMware CIM SDK, developers can use CIM-compliant applications to explore the virtual machines on ESX Server, along with associated storage resources.

5.2 Grid information schema and information service

Various types of resources which are shared on computational Grids should be described in a precise and systematic manner. The Grid resources are thus able to be discovered for subsequent management or use.

The GLUE (Grid Laboratory Uniform Environment) schema [13] represents an abstract model for Grid resources and mappings to concrete schemas that can be used by information services within Grids. The underlying idea of the schema therefore is to provide an information model that can be used to exchange pieces of information among different knowledge domains and virtual organizations [14]. The GLUE schema is widely used in production Grid such as EGEE [10], OSG [18] and Teragrid [25].

The GLUE schema is defined in UML diagrams (1.3 version) or XML (1.2 version). The GLUE schema defines so called core entities, such as `Site`, `Service`, `ComputingElement` and `StorageElement`. The relations between core entities are represented in concept level, such as objects and properties.

An Grid information service provides information about a Grid infrastructure that consists of a wide variety of Grid resources. Grid information is thereafter used for various Grid operations, such as resource discovery/monitoring/accounting and job submission/execution.

In our production Grid environment, the Globus index service is used for the Grid level information service. The Grid index service can collect information and publish the information to clients. The Grid index service can also register to each other in a hierarchical fashion in order to aggregate data at several levels. The `Aggregator Framework` of Globus index service is used to build services that collect and aggregate data. For example, we build an information provider from virtual machines and provide information to the `Aggregator Framework`.

5.3 Translation of CIM information to GLUE information

The virtual machine information is organized as a CIM schema. In production computational Grids, resource information is defined by GLUE schema. The information provider therefore needs to translate the CIM information into GLUE information, then marshes the data into GLUE XML file.

There are mainly two technical problems to translate CIM information to GLUE information:

- The GLUE schema does not contain virtual machine concept.
- CIM schema and GLUE schema make their own representation at different levels. It is thus impossible to make a direct mapping of CIM schema to GLUE schema.

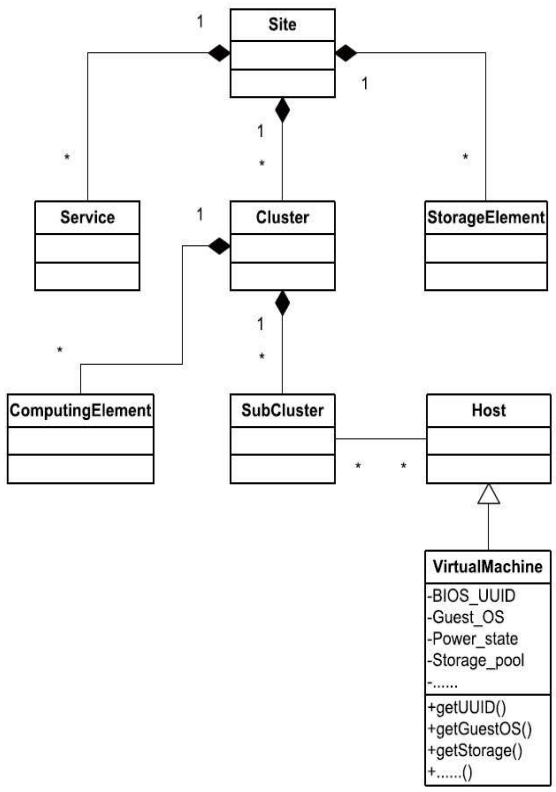


Figure 6. Extended GLUE schema

The GLUE schema has been extended accordingly to support virtual machine concept. A **VirtualMachine** class is created by inheriting **Host** class in the GLUE schema. Several attributes and operations are included in **VirtualMachine** classes to represent virtual machine concepts. Figure 6 shows the extended GLUE schema.

The GLUE schema is represented in various high level formats, such as XML schema and UML class diagram. The CIM schema, on the other hand, provides an object oriented format. The CIM schema shipped by VMware is represented in Java classes. Thus there is no direct mapping from CIM schema to GLUE schema. We first generated Java classes from GLUE schema. The Java Architecture for XML Binding (JAXB) provides a fast and convenient way to bind an XML schema to Java representations. With the JAXB compiler and binding tools, Java binding classes could be automatically generated from GLUE schema. In the binding classes, CIM GLUE classes (Pegasus CIMON APIs) are invoked and virtual machine information is generated. The information is thereafter marshaled to GLUE

XML data.

6 Information Provider

An information provider has been programmed to call the information collector client and the translator automatically and read the GLUE XML file to the Globus Aggregator Framework.

It is also required to configure Globus `Aggregator Framework` to enable the information provider for Globus index service, for example, registering the information provider to the back-end of Globus index service, mapping the information provider in the deployment file of the Globus index service and configure the schedule to run the information provider. Therefore GLUE XML data could be automatically generated and provided to Globus `Aggregator Framework`. The Grid index service could provide resource information from virtual machines to higher Grid services, for example, WebMDS [28] or be accessed from the Globus Toolkit command line.

7 Test results

7.1 Test bed

The actual test bed is configured as shown in Fig. 7. **Blade10**, **Blade11** and **Lizhe3** are hosting resources which are installed with VMware ESX server and VMware server. **VM1**, **VM2** and **VM3** are virtual machines backed by the hosting resources that form the virtual machine pool. **Lizhe2** is the access point for the virtual machine pool (see also Tab. 2).

7.2 Test results

The access point runs an information provider to retrieve information from **VM1**/**VM3** and **VM2** via VMware ESX server. The information provider collects the resource information and organizes it with GLUE schema (shown in Tab. 1)

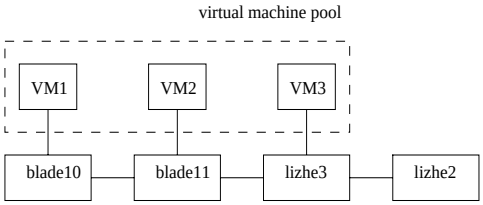


Figure 7. Test bed

The information provider thus furnishes the organized information to the `Aggregator Framework` of Globus Toolkit. Users can retrieve the information with client programs of Globus index service or browse the information via WebMDS. Fig. 8 shows the resource information retrieved from virtual machine pool with WebMDS on **Lizhe2**. These results justify the correct operation of the prototype implemented on the test bed.

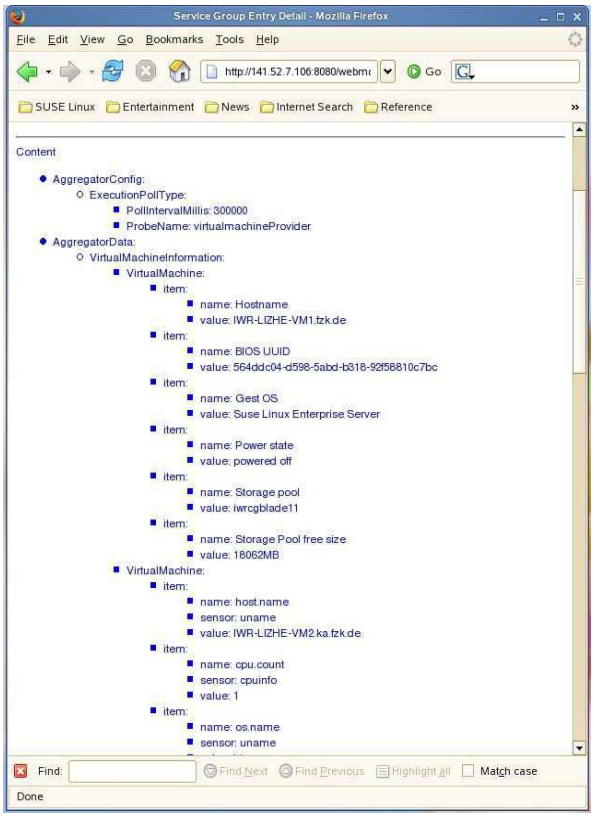


Figure 8. Resource information from virtual machines

Table 1. Virtual machine information in GLUE schema

```
< VirtualMachineInformation >
  ...
  < VirtualMachine >
    < itemname = "Hostname" value = "IWR − LIZHE − VM2.fzk.de" / >
    < itemname = "BIOS UUID" value = "564ddc04 − d598 − 5abd − b318 − 92f58810c7bc" / >
    < itemname = "Guest OS" value = "Suse Linux Enterprise Server" / >
    < itemname = "Power state" value = "powered off" / >
    < itemname = "Storagepool" value = "iwrcgblade11" / >
    ...
  < /VirtualMachine >
  ...
< /VirtualMachineInformation >
```

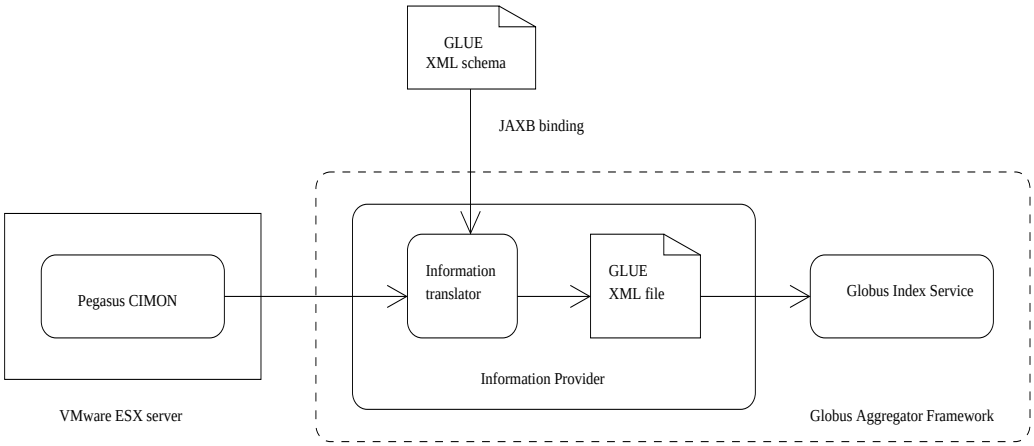


Figure 9. Implementation of virtual machine information provider

Table 2. Test bed summary

Resource Name	Resource Type	Software installed
Blade9	IBM BladeCenter HS20 2× Intel© Xeon^{TM} CPU 3.00 GHz	VMWare ESX server
Blade11	IBM BladeCenter LS20 2×AMD Opteron^{TM} Processor 250, 2.8 GHz	VMware ESX server
IWR-Lizhe3	Linux Workstation 1× AMD Athlon^{TM} 64 Processor 3500 +	VMware server
IWR-Lizhe2	Linux Workstation 1×Intel© Pentium M Processor 1.5GHz	Globus Toolkit 4
VM1, VM2, VM3	Virtual Machine	Scientific Linux 3.0.6

8 Conclusion and Future work

Virtual machines are widely accepted in computer centers to support various applications. This paper implements an information service of virtual machine pools for Grid computing. The information service can monitor virtual machines backed by popular VMM, such as VMware ESX server. We argue our contributions are

- building an efficient information service for retrieving virtual machines, and
- translating CIM based virtual machine information to Grid information defined in GLUE schema.

This is the first attempt of bridging the gap between industry information standard and production Grid information standard.

The prototype of the implementation will be developed and tested in large scale Grid system, e.g. D-Grid test bed [8]. The performance such as scalability and robustness should be considered for further implementation.

References

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From Virtualized Resources to Virtual Computing Grids: the In-VIGO System. *Future Generation Computing Systems*, 21(6):896–909, June 2005.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization . In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, New York, USA, Oct 2003. ACM Press.

[3] V. Buege, Y. Kemp, M. Kunze, O. Oberst, and G. Quast. Virtualizing a Batch Queueing System at a University Grid Center. *Proceeding of Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC), LNCS*, 4331:Italy, 397-406 2006.

[4] S. Childs, B. Coghlan, and J. McCandless. GridBuilder: A Tool for Creating Virtual Grid Testbeds. In *Proceedings of 2nd IEEE Conference on eScience and Grid computing (e-Science)*, pages 77–77, Amsterdam, Netherlands, Dec. 2006. IEEE Computer Society.

[5] S. Childs, B. Coghlan, D. O'Callaghan, G. Quigley, and J. Walsh. A Single-computer Grid Gateway Using Virtual Machines. In *Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, pages 310 –315, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Common Information Model (CIM). http://www.dmtf.org/standards/cim/, 2005.

[7] Pegasus CIMOM. http://www.openpegasus.org.

[8] D-Grid project. http://www.d-grid.org.

[9] N. Fallenbeck, H. J. Picht, M. Smith, and B. Freisleben. Xen and the art of cluster scheduling. In *Proc. of 1st International Workshop on Virtualization Technology in Distributed Computing*, USA, Nov. 2006. IEEE Computer Society.

[10] Enabling Grids for E-sciencE (EGEE). http://www.eu-egee.org/.

[11] Dsitributed Management Task Force. http://www.dmtf.org.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Philosophy of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Open Grid Service Infrastructure Workgroup, Global Grid Forum, 2002.

[13] Open Grid Forum Glue Schema Working Group. Glue schema v1.3, Jan 2006.

[14] Open Grid Forum Grid Interoperation Now Community Group. Experiences from interoperation scenarios in production grids, Feb. 2007.

[15] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual Workspaces in the Grid. *Proceedings 11th International Euro-Par Conference, LNCS*, 3648:421–431, 2005.

[16] User Mode Linux. http://user-mode-linux.sourceforge.net.

[17] University of Cambridge Computer Laboratory. Performance comparison of vmms. available from http://www.cl.cam.ac.uk/research/srg/netos/xen/performance.html, July 2006.

[18] Open Science Grid (OSG). http://www.opensciencegrid.org/, 2007.

[19] P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Towards Virtual Distributed Environments in a Shared Infrustructure. *IEEE Computer*, 38(5):63–69, 2005.

[20] S. Soltesz, M. E. Fiuczynski, L. Peterson, M. McCabe, and J. Matthews. Virtual Doppelganger: On the Performance, Isolation, and Scalability of Para- and Paene- Virtualized Systems.
available from http://www.cs.princeton.edu/∽mef/research/paenevirtualization.pdf, Nov. 2005.

[21] Storage Management Initiative Specification.
http://www.snia.org/smi/tech_activities/smi_spec_pr/spec.

[22] P. Strazdins, R. Alexander, and D. Barr. Performance Enhancement of SMP Clusters with Multiple Network Interfaces Using Virtualization. *Proceeding of Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC), LNCS*, 4331:452–463, 2006.

[23] Ananth I. Sundararaj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *Virtual Machine Research and Technology Symposium*, pages 177–190, 2004.

[24] M. Tatezono, N. Maruyama, and S.Matsuoka. Making Wide-Area, Multi-Site MPI Feasible Using Xen VM. *Proceeding of Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC), LNCS*, 4331:387–396, 2006.

[25] Teragrid. http://www.teragrid.org/, 2007.

[26] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Wang. Seamless Live Migration of Virtual Machines over the MAN/WAN. *Future Generations Computer Systems*, 22:901–907, 2006.

[27] VMware virtualization products.
http://www.vmware.com.

[28] WebMDS.
http://www.globus.org/toolkit/docs/4.0/info/webmds.

Virtualization of Grid Services in D-Grid

F. Kulla, M. Kunze
Institut für Wissenschaftliches Rechnen, Forschungszentrum Karlsruhe
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
{Fabian.Kulla, Marcel.Kunze}@iwr.fzk.de

Abstract

Grid services need to be operated in a reliable and scalable way in order to guarantee round the clock access to resources for a widely distributed user community. Virtualization techniques seem to be well suited to set up highly available automated systems. Bringing both worlds together, this paper discusses an implementation to run virtualized Grid services for the German D-Grid infrastructure.

1. Introduction

The D-Grid initiative aims to build the foundation for a sustainable Grid infrastructure in Germany [1]. The D-Grid integration project works on the provisioning of generic Grid middleware components and the operation of common sustainable Grid services for the D-Grid communities (Fig. 1).

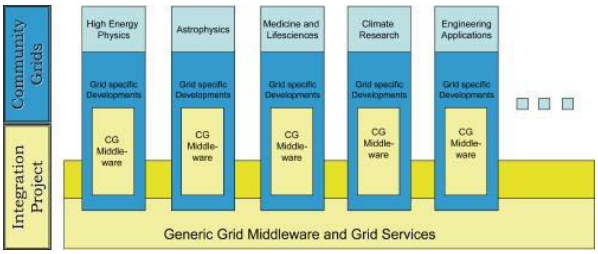


Fig.1: D-Grid community projects are supported by the integration project that supports generic Grid middleware and Grid services.

The middleware architecture is based on three independent flavours: Globus Toolkit [2], gLite [3] and Unicore [4]. The generic layer requires not only the operation of middleware specific services like resource broker, scheduler or resource monitor but as well the implementation of gateways to route requests between the different Grid worlds and translate the corresponding information.

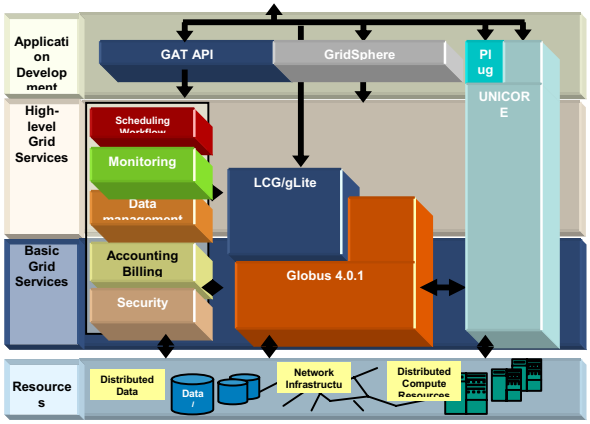


Fig.2: The D-Grid middleware architecture builds on Globus Toolkit, gLite and Unicore.

It is the task of Forschungszentrum Karlsruhe to host the numerous basic Grid services in the D-Grid integration project and deliver reliable operations. The usual procedure for robust operation foresees to host services on individual servers in order to isolate the individual tasks and in addition have less interference during the configuration of e.g. network ports and software libraries. This procedure, however, needs a considerable amount of resources that are very often underutilized as the individual services may not be constantly on high load. In addition, service availability problems arise in the case of server maintenance or server downtime. This is a serious issue as the basic Grid services are essential for all Grid applications and have to be operated round the clock in a widely distributed environment. It is thus mandatory to guarantee high availability and to be able to define and monitor service level agreements.

2. Virtual Infrastructure

In order to solve the problems mentioned above we decided to use virtual machines to implement the Grid services and perform all operations based on a generic virtual infrastructure. An analysis of requirements was performed taking into account the following aspects:

- Management tools
- Resource pooling
- Provisioning of machines
- Service level definition
- Template libraries
- Snapshots and backup
- Live migration of machines
- Monitoring capabilities
- Virtual LAN to access machines remotely
- Virtual SMP to support multi-core
- Virtual memory efficiency

The evaluation was done in view of the establishment of a high availability solution that allows to group resources in resource pools.

We evaluated three different virtualization solutions: Microsoft Virtualserver [5], VMware Virtual Infrastructure [6], and Citrix XenServer [7]. The various products come with different management capabilities, and are architected in a multi-tier structure: There is usually a bulk of physical servers to host the virtual machines and a management server that coordinates the resource pools and takes care of resource usage. The steering and control is performed on a further level either using a dedicated console or by use of a Web interface.

Considering the requirements the decision was to set up a high availability cluster on the basis of IBM BladeCenter, bundled with VMware Virtual Infrastructure (see Fig. 3). The cluster consists of two identical BladeCenter systems, each equipped with 7 quad-core blades HS21 with 16 GB RAM that are installed in two geographically separated data centres on the campus of Forschungszentrum Karlsruhe. The storage pools contain around 10 TeraBytes disk space and have been realized using a distributed setup in a common storage area network (SAN). Special care has been taken to implement all features and network connections to allow for redundancy and smart failover. The network components in Figure 3 are coloured light blue. Every blade server has four network interfaces which are operated in active teaming mode in order to gain performance and fault tolerance. Two of them are connected to one switch and the others are connected to the second switch. The uplink ports of these switches are connected to different backbone switches in a way that network failures can be compensated. All resources like storage, CPU cycles, memory, networking bandwidth are organized in resource pools, independent of their location, and can dynamically be assigned to the virtual machines hosting the Grid services.

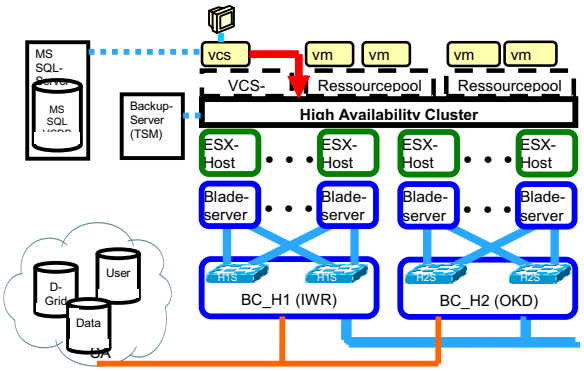


Fig. 3: Virtual infrastructure: High availability cluster, distributed over two data centres (IWR and OKD). All management information is maintained in a SQL database. Virtual machine snapshots are backed up in a TSM tape library.

A high availability cluster has been arranged to host the nearly 50 Grid service machines in a way such that half of the resources are located in each data centre. The corresponding management services are run on the basis of the VirtualCenter management server and console. It is worthwhile to mention that we implemented the management server itself as a virtual machine inside the high availability cluster in order to gain reliability. This in-band solution is feasible as the server only takes care of planning and monitoring tasks; the operations are executed by the ESX blades autonomously by high availability agents. It can thus be guaranteed that missing virtual machines are restarted even in absence of the management server.

Virtual machines are backed up to a TSM tape service on user demand or in a fully automated rule based fashion using Vizioncore vRanger [8]. The backup works with virtual machine snapshots that are taken on-line. These snapshots are gathered on a dedicated Windows blade server from where they are moved to a tape library.

As the resources are always accessed through the virtualization layer, it is possible to define service levels to be monitored in the VirtualCenter console. In order to maintain service levels, the management server automatically moves the virtual machines across the physical layer depending on a load balancing mechanism. In case of system or firmware upgrades individual physical hosts can enter maintenance mode. They are drained without implications for the virtual machine operation, as the load can be moved to other active resources in the cluster. Similarly, additional blade capacity can be dynamically added to the cluster in case of performance bottlenecks. In case of a sudden system or power failure in one of the computer centres the missing services are supposed to automatically restart using the remaining resources.

The virtual machine storage is partitioned out of the common SAN storage pools by definition of LUNs that are mounted as SCSI devices at each blade. The VMware virtual machines file system (VMFS) safely grants concurrent and efficient access to the disks even in larger installations. The system allows for migration of machine storage from one LUN to the other during operation. This feature enables replacement of storage systems without any downtime.

Fig.4 shows a snapshot of the VirtualCenter management console with the virtual machines deployed for D-Grid.

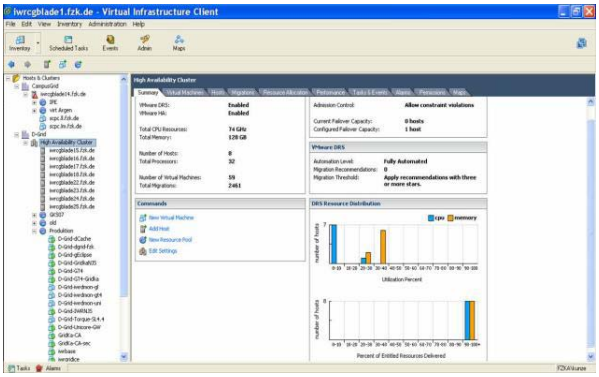


Fig.4: Management console of the high availability cluster hosting the D-Grid virtual machines. Shown is the main window with tabs to navigate to different views. Left pane: Resource and machine overview. Lower right: Monitoring of the service levels as delivered by the high availability cluster.

The authentication scheme to manage the virtual servers is based on Microsoft Active Directory. Using VirtualCenter, roles like administrator, user, developer etc. can be defined and all settings are maintained in a SQL database. According to the specific roles management operations can be allowed or forbidden on the basis of fine-grained resource access rules. This security feature is essential for running common resource pools in a multi-domain environment and has to be compared to the Xen world where each management activity requires superuser privileges.

3. Discussion

The system described above is in reliable operation since more than a year. From our practical experience there is lot of added value from virtualization as compared to a static server setup. The most important point is that virtualization eases the way to data centre automation through the concept of resource pool management. The main advantages are:

- Rapid deployment of virtual servers
- Automated load balancing
- Automated failover
- Performance guarantees
- Snapshots of systems
- Support of legacy systems

In total we observe an increased flexibility to satisfy customer requirements at a lower cost, as we have less maintenance effort due to the standardized and fault tolerant physical environment. However, one has to take into account that there is additional effort to maintain the virtualization layer and that there is a need to install high-quality infrastructure on the physical layer (e.g. SAN, multi-core servers with maximum memory).

The server consolidation leads to a better utilization of the hardware and thus decreases energy consumption considerably. The consolidation factor is in the range of 5 to 20 virtual machines per physical server blade, depending on the actual load. In times of low utilization, machines may be concentrated on a few blades and empty blades switched off in order to lower energy consumption. In case of a need, the system is able to re-activate blades automatically by Wake-On-LAN calls in order to increase the capacity of the pool.

An additional set of machines has been arranged in a testing pool at lower service levels. Furthermore, the virtual infrastructure had also been used at the same time to host another 50 virtual machines to train the 140 students during the week of GridKa school 2007

[9]. The training courses used Thin Clients hooked up to the virtual machines, a concept that actually seems very promising for reasons of flexibility and economy [10]. On the other hand the use of virtual networks allows virtual machines to reside in the subnet of distant users, opening the virtual floor to as well support application specific areas. Dedicated hardware in remote institutes could thus be supplemented by compute power out of the centralized common resource pool.

It is our observation that the performance penalties through the virtualization layer are in the order of 5% only and thus almost negligible. Developers have even reported a higher performance as compared to their local PC based workstations. This is due to the better I/O service levels that can be arranged in the SAN based storage and an easy to deliver larger RAM allocation.

Taking snapshots of systems is of great benefit for developers, as they can return to a specific point of their work at any time. Arbitrary system changes can be accepted or withdrawn with just a mouse-click.

Of special value is the support of legacy systems: It allows us to maintain 32 bit applications on actual 64 bit hardware. This enabled us to easily migrate elder middleware components to the most recent powerful multi-core blade systems without major problems.

4. Outlook

It is envisaged to considerably increase the blade capacity by further 10 chassis for D-Grid (some 1200 cores). These blades will be equipped with Infiniband interconnect and will serve as a resource for high performance parallel computing. Besides VMware Virtual Infrastructure we envisage to use OpenSource Xen to implement and manage virtual systems. Another area of current activities is the evaluation of storage virtualization solutions to make better use of storage devices and for instance allow for thin provisioning of capacity.

The high availability concepts described in this paper are as well interesting for delivery of high quality in-house services. We intend to deploy another high availability cluster over a distance of 12 km between Forschungszentrum Karlsruhe and Karlsruhe University as a foundation for the services to be delivered to the newly formed Karlsruhe Institute of Technology (KIT).

10. References

[1] The D-Grid initiative, http://www.d-grid.de/
[2] The Globus Toolkit, http://www.globus.org/
[3] gLite, http://glite.web.cern.ch/glite/
[4] Unicore, http://www.unicore.eu/
[5] Microsoft Virtual Server, http://www.microsoft.com/windowsserversystem/virtualserver/
[6] VMware Virtual Infrastructure, http://www.vmware.com/products/vi/
[7] Citrix XenServer, http://www.citrixxenserver.com/Pages/default.aspx
[8] Vizioncore vRanger Pro, http://www.vizioncore.com/
[9] GridKa School 2007, http://gks07.fzk.de/
[10] C. Knermann, PC vs. Thin Client, 2007, http://it.umsicht.fraunhofer.de/PCvsTC/index_en.html

Management

Towards a Framework for the Autonomic Management of Virtualization-Based Environments

Dan Marinescu and Reinhold Kroeger
Wiesbaden University of Applied Sciences
Distributed Systems Lab
Kurt-Schumacher-Ring 18, D-65197 Wiesbaden, Germany
{*dan.marinescu,kroeger*}*@informatik.fh-wiesbaden.de*

Abstract

Over the past years, virtualization has emerged again, making its place in the data centers. However, data centers are already facing management problems due to the increased system complexity. Virtualization increases this complexity even more. To address these management problems, computing systems should be able to manage themselves. In this paper, we propose a modular framework for the autonomic management of virtualization-based environments. This framework is virtualization-technology independent and permits plugging in various controller approaches.

1. Introduction

System virtualization is a technique first developed in the mid 1960's. It consists of an indirection layer that is introduced between the hardware and the operating system, called virtual machine monitor (VMM) or hypervisor. The VMM partitions the hardware in logical units, called virtual machines (VMs). Inside a VM, a so-called guest operating system is running. The VMM controls and synchronizes the access of guest OSs to hardware resources. As such, it is possible to run multiple, possibly different OSs in parallel on the same hardware. Although a flourishing technology in the 1970's, the 1980's and 1990's brought a drop in hardware prices which meant that it was affordable to run one application per computing system, and thus virtualization was slowly forgotten [10]. Over the past years howvever, virtualization has emerged again, nowadays being used on both server (e.g. VMware ESX [19], Xen [23]) and desktop systems (e.g. VMWare Workstation [20], Virtual Box [18]). The new paradigm states: one application per virtual machine, several virtual machines per computing system.

The adoption of virtualization in the data center is taking place at a high pace. The possibility of consolidating a bundle of under-utilized server boxes into one server system is certainly one of the reasons. Besides server consolidation, it is the way maintenance and system management are simplified that makes this technology attractive for system administrators. However, virtualization does not reduce the complexity of a system. In fact, having multiple virtual machines running on top of several physical machines actually increases the overall system complexity. This matches the theory of the ever increasing complexity of computing systems, which will finally lead to systems which cannot be managed by human experts (system administrators) any more. However, such systems exist in the nature since millions of years. In [6], the authors proposed the human autonomic nervous system (ANS) as an inspiration source. The ANS manages low-level, yet vital body functions like heart rate or body temperature without conscious control. Similarly, computing systems should be able to manage themselves, based on some high-level goals defined by a human system administrator. The field of reseach that deals with this kind of systems is called autonomic computing, the essence of which is self-management. According to [6], there are four aspects of self-management:

- self-configuration, the capability of computing systems to install and configure themselves
- self-optimization, the property of computing systems to continuously seek to improve their operation
- self-healing, the property of computing systems to detect, diagnose and repair local problems
- self-protection, the property of computing systems to defend themselves in the face of a malicious attack or cascading failures

For the purpose of this paper, we will mainly focus on the self-optimization aspect for the management of

virtualization-based environments, assuming that quality-of-service levels of the services, provided by the VMs, have to be fulfilled [3].

Virtualization-based environments are a great area of application for autonomic computing. On one hand, the increased complexity caused by the use of virtualization makes autonomic computing a necessity in large data centers. On the other hand, most of the server virtualization technologies that currently exist provide management features like the allocation of resources (memory, CPU shares and others) at run-time and live migration of virtual machines [2]. These features are great control knobs (actuators) that can be used for system self-optimization. Also monitoring and debugging capabilities are enhanced by the introduction of an indirection layer between the operating system and the hardware. What is missing is an intelligent controller that can take decisions based on the data gathered through monitoring resulting in dynamically allocated resources through the previously mentioned actuators. This paper deals with aspects related to the development of such an intelligent controller.

2. Related work and limitations

Various publications deal with the autonomic management of virtual environments. Policy-based approaches have been a popular way to manage computing systems, and virtualization-based systems are no exception. In [12], the authors present a system called VIOLION, which uses the Xen hypervisor for virtualization. The system comprises one monitor daemon per physical machine and one adaptation manager. The adaptation manager uses the data gathered by the monitor daemon to dictate virtual machine resource allocation. Another policy-based approach has been published by Grit et al. [4]. Here, the authors use and extend Shirako, a Java-based toolkit for resource leasing services, to explore algorithmic challenges when using policies for the adaptive hosting of virtual machines on computer clusters.

Control theory has also been used to autonomically manage virtual environments. Zhang et al. [24] present a control-theoretic model for VM adaptation, based on the idea that the VMs are responsible for adjusting their demand for resources, with respect to efficiency and fairness. In [9], the authors use a feedback-control strategy to address dynamic resource allocation problems. The approach uses an infrastructure based on Xen, RUBiS [11] and TPC-W [16]. Time series analysis can be used to forecast the behaviour of a virtualization-based system. Bobroff et al. [1] present a mechanism for the dynamic migration of virtual machines based on the forecasted load. Menascè et al. [8] use an approach based on utility functions [21] for the dynamic CPU allocation to virtual machines. The authors test their approach by means of simulations, using historical data.

These publications represent the first steps taken by the research community to develop various strategies for an intelligent controller based on different control paradigms. It is however hard to objectively evaluate these strategies. The various publications use different architectures, perform different tests and some even only use historical data and simulation to test how their strategies perform. In this context, it is impossible to say that one approach performs better than another, or that some approach is more adequate in a specific situation than the other approaches. The main reason for this is that automated management of VM environments is not mature yet but still in its infancy. We argue that the only way to solve this problem is to design a common framework for the development and evaluation of autonomic computing strategies for virtualization-based distributed environments.

3. Architecture of the framework

A framework for developing and evaluating autonomic computing strategies for virtualization-based distributed environments must fulfill the following requirements:

1. Support for service level management
2. Separation of control algorithms from the management framework
3. Support for different virtualization technologies
4. Support for a common evaluation mechanism
5. Scalability

(1) means that the framework should provide support for managing services hosted in virtual machines. This involves monitoring various QoS parameters and making sure that these parameters conform to a certain Service Level Objective (SLO).

(2) basically assures that the management framework is responsible for dealing with aspects like monitoring and execution of tasks, while a separate intelligent controller defines management tasks based on gathered monitoring data. As such, a framework should support different types of intelligent controllers by providing a generic controller interface.

(3) requires the framework to support different virtualization technologies (e.g. VMware ESX or Xen), transparently for the controller. Thus, from a controller's perspective, the framework is acting as an abstraction layer on top of the virtualization technology. Generic interfaces for monitoring and task execution should simplify plugging in adapters for various virtualization technologies.

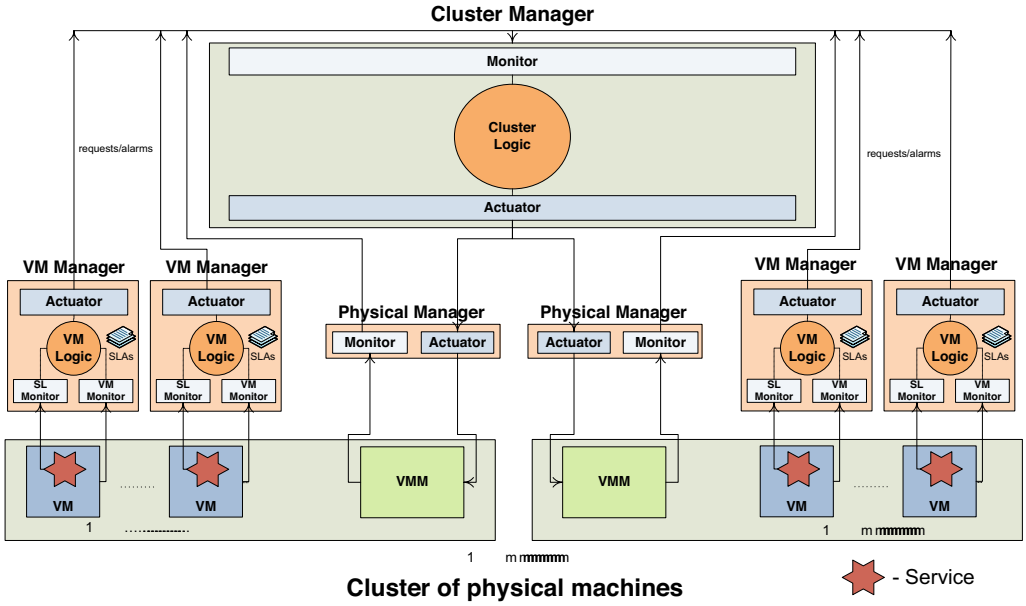


Figure 1. The architecture of the framework

(4) means that the framework has to support a consistent mechanism for tracing and comparing management decisions. Only this way different self-management approaches can be evaluated against each other.

(5) states that the framework should be scalable with respect to the number of physical and virtual machines that it can manage. New physical or virtual machines dynamically added to the cluster must be recognized by the framework.

Figure 1 depicts the design of our framework with respect to the requirements discussed above. The framework is used to manage a cluster of n physical machines, each hosting a number between 0 and m virtual machines. Each virtual machine has a *VM Manager*, each physical machine has a *Physical Manager* and the entire cluster is managed by one *Cluster Manager*.

The Physical Manager is the simplest component of the framework. It's job is to monitor the resource utilization of the physical machine through its *Monitor* module. Besides monitoring, the Physical Manager also executes commands coming from the Cluster Manager through its *Actuator* module. Both the Monitor and the Actuator modules monitor respectively control the physical machine through the *VMM*.

The VM Manager monitors parameters like CPU utilization and available memory with respect to the virtual machine through the *VM Monitor* module. It also monitors the service/application running inside the virtual machine through the *SL Monitor* module. We assume that each virtual machine hosts only one service, e.g. a web server, a mail server or a database; we argue that this is common practice in a server consolidation scenario. The two monitors feed the gathered data to the *VM Logic* module, which uses this data to determine whether the service is running at the required parameters. If this is not the case, the VM Logic tries to determine which of the VM's resource is the bottleneck (i.e. memory, CPU, etc.). When such a bottleneck is determined, the VM Manager uses the *VM Actuator* module to request a larger amount of the resource that causes the bottleneck from the Cluster Manager (i.e. request more memory, a larger CPU share etc.).

The Cluster Manager has a global view of both available and allocated resources, generated using the data gathered from the Physical Managers. The *Cluster Logic* module uses this global view to determine how requests from the VM Manager can be fulfilled. Thus, the Cluster Logic module is actually what we previously referred to as the intelligent controller. After a decision has been taken, the Cluster Manager dictates commands to the Physical Manager(s) through its *Actuator* module.

It can be easily observed that this framework fulfills the previously defined requirements. First, the way that the VM Manager is designed specifically addresses the service running in the "managed" virtual machine. Various Cluster Logic components that use different self-management strategies can be easily plugged into the framework. Furthermore, different virtualization technologies can be addressed by simply developing the necessary adapters for each virtualization technology. It is possible to use stan-

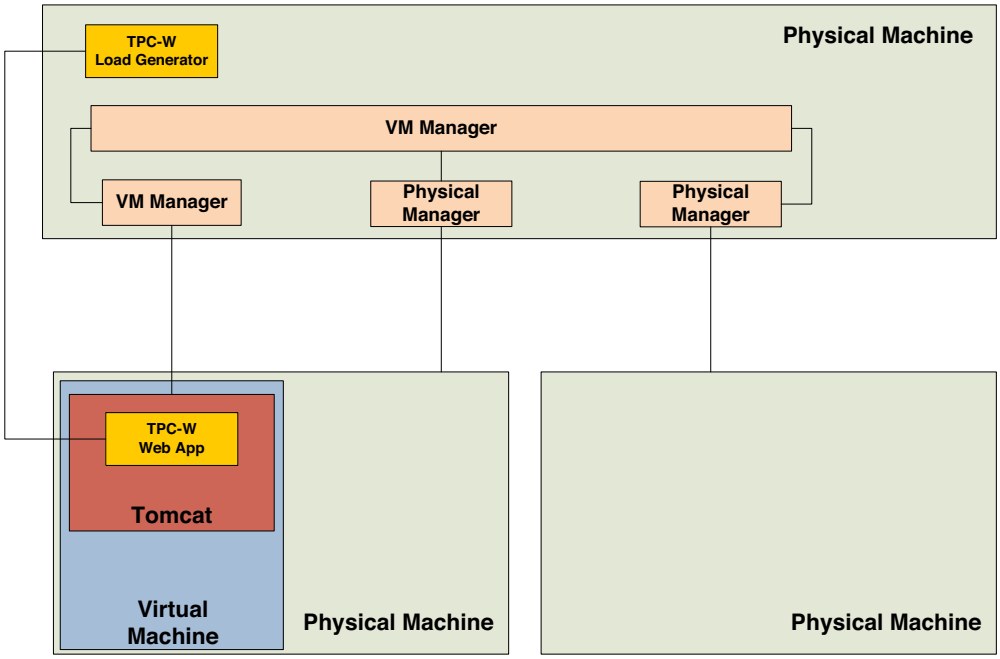


Figure 2. The testbed

dard benchmarking tools to generate load for the services running inside the virtual machines and then use the VM Monitor component to evaluate how these services are performing. This way, different intelligent controllers can be evaluated with respect to how the services are performing under various loads. Last, adding new physical and virtual machines to the cluster can be done by simply registering with the cluster manager.

4. Implementation

We have implemented a prototype of the framework in Java using Xen as the virtualization technology. Our prototype implementation is based on the Self Manager Core that has been developed in the Distributed Systems Lab of Wiesbaden University of Applied Sciences [15, 14]. The VM Manager uses JMX [5] to monitor the parameters of a Xen-based virtual machine (the VM Monitor component) and the response times of an Apache Tomcat server running inside the virtual machine (the SL Monitor component). The VM Logic is implemented as an expert system and uses the data from the SL Monitor and the VM Monitor to feed Jess [13], a rule engine. The rule engine matches the data against a rule set. We have developed a basic set of rules, which is used by Jess to determine the bottlenecked resource when the response times of the service are higher than a given threshold. For the Physical Manager, an adapter has been implemented that uses the Xen-API [22] to manage Xen-based physical machines. This involves both monitoring and dynamically allocating resources. Both the VM Manager and the Physical Manager communicate with the Cluster Manager using Java RMI. The Cluster Manager provides support for plugging-in different Cluster Logic components through a messaging interface. More implementation details can be found in [7].

We have tested the framework in a lab environment using a Java implementation of the TPC-W benchmarking standard [17]. This testbed can be observed in Figure 2. The TPC-W e-commerce application suite is hosted inside a Tomcat server running on top of a virtual machine managed by the VM Manager. Two physical machines are managed by two different Physical Managers, with one of them hosting the virtual machine. The two Physical Managers, the VM Manager and the Cluster Manager are hosted on a third physical machine, which also hosts the TPC-W load generator. This machine is not part of the "under-management" cluster. When the TPC-W load generator is started, the VM Manager is able to observe a dramatic increase in the response times of the Tomcat service and determines the memory as being the bottleneck. After more memory is given, the response times of the Tomcat service decrease up to an acceptable level. This behaviour is shown in Figure 3 and proves that the VM Manager is able to successfully determine the resource causing the bottleneck. Distributed

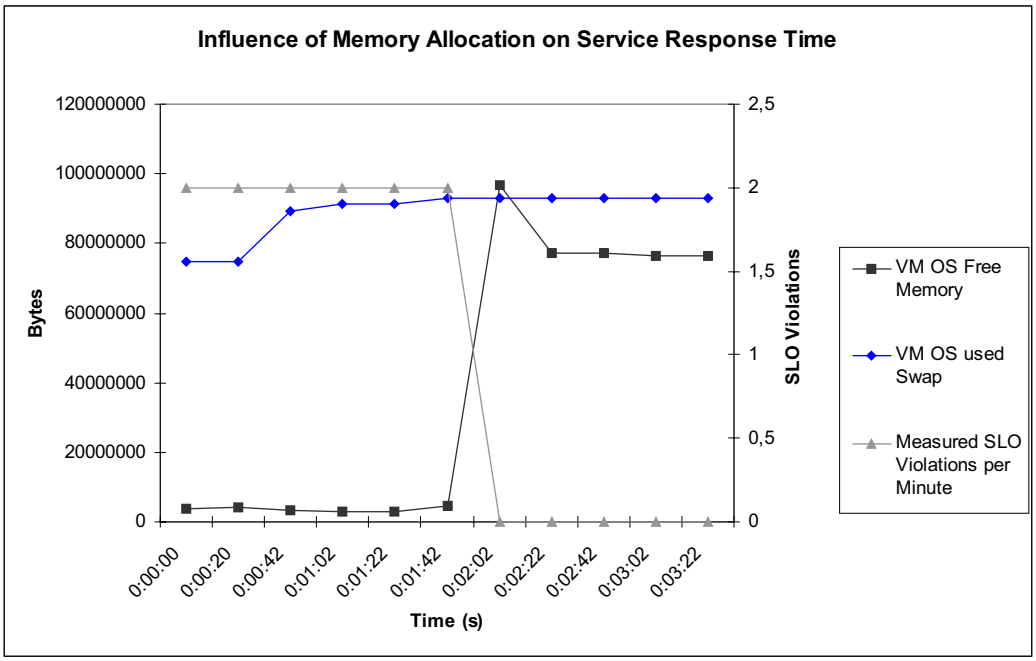


Figure 3. Influence of available memory on SLO violations

tests including migrating a VM on a secong physical machine when necessary are currently underway.

5. Conclusion and outlook

In this paper, we have argued that virtualization-based environments are an important application area for autonomic computing, but still in its infancy at the moment. Furthermore, it is hard to objectively evaluate the currently published autonomic computing approaches for virtualization-based environments. The main contribution of this paper is the design of a framework for developing and evaluating autonomic computing strategies for virtualization-based environments. Using this framework, different controller strategies can be easily developed, plugged-in and evaluated against each other. Our framework is not virtualization technology-bound, instead adapters for various virtualization technologies can be easily developed and plugged into the framework.

We are currently working on different approaches for an intelligent controller. We have discovered that our resource allocation problem belongs to the family of knapsack problems. Knapsack problems are known to be NP-hard. This means that we are dealing with an NP-hard optimization problem. This is an important discovery, since it is commonly believed that, for NP-hard optimization problems, no algorithm exists that finds an optimal solution in polynomial time. As such, for our resource allocation problem, we can either use heuristics or approximation algorithms. While for the moment we are working on two different heuristic-based algorithms, in the future we plan to design an intelligent controller that uses an approximation algorithm and thus has a guaranteed performance ratio. Details about the design and evaluation of these controllers as a Cluster Logic component in our framework will be made available in a future publication.

References

[1] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, 2007.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286, Boston, MA, May 2005.

[3] M. Debusmann, M. Schmid, and R. Kroeger. Model-Driven Self-Management of Legacy Applications. In L. Kutvonen and N. Alonistioti, editors, *5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece, June 2005*, pages 56–67. IFIP, Springer, June 2005.

[4] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase. Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration.

[5] Java Management Extensions. http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/ Last visited 11.12.2007.

[6] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, 2003.

[7] D. Marinescu. Design and evaluation of self-management approaches for virtual machine-based environments. Master's thesis, Wiesbaden University of Applied Sciences, DCSM, February 2008.

[8] D. A. Menasce and M. N. Bennani. Autonomic virtualized environments. In *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*, page 28, Washington, DC, USA, 2006. IEEE Computer Society.

[9] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *EuroSys '07: Proceedings of the 2007 conference on EuroSys*, pages 289–302, New York, NY, USA, 2007. ACM Press.

[10] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, 2005.

[11] RUBiS. http://rubis.objectweb.org/ Last visited 11.12.2007.

[12] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 5–14, 2006.

[13] Sandia National Laboratories. http://herzberg.ca.sandia.gov/ (Last visited 31.01.2008).

[14] M. Schmid. Ein Ansatz fuer das Service Level Management in dynamischen Architekturen. In T. Braun, G. Carle, and B. Stiller, editors, *KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeitraege, Kurzbeitraege und Workshops*, pages 255–266. VDE Verlag, March 2007.

[15] M. Schmid and K. Geihs. Self-Organisation in the Context of QoS Management in Service Oriented Architectures. In K. Boudaoud, N. Nobelis, and T. Nebe, editors, *Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by University of Nice at Cote d'Azur May 21 - 24, 2006*, pages 153–164, Stuttgart, May 2006. Infonomics-Consulting.

[16] TPC-W. http://www.tpc.org/tpcw/ Last visited 11.12.2007.

[17] TPC-W Java Implementation. http://www.ece.wisc.edu/~pharm/tpcw.shtml Last visited 11.12.2007.

[18] Virtual Box. http://www.virtualbox.org/ Last visited 11.12.2007.

[19] VMware ESX. http://www.vmware.com/products/vi/esx/ Last visited 11.12.2007.

[20] VMware Workstation. http://www.vmware.com/products/ws/ Last visited 11.12.2007.

[21] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 70–77, 2004.

[22] The Xen API. http://wiki.xensource.com/xenwiki/XenApi Last visited 11.12.2007.

[23] Xen Source. http://www.xensource.com/ Last visited 11.12.2007.

[24] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West. Friendly virtual machines: leveraging a feedback-control model for application adaptation. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 2–12, New York, NY, USA, 2005. ACM Press.

Fault and Intrusion Tolerance on the Basis of Virtual Machines *

Hans P. Reiser
LaSIGE
Universidade de Lisboa
Portugal
hans@di.fc.ul.pt

Rüdiger Kapitza
Informatik 4
University of Erlangen-Nürnberg
Germany
rrkapitz@cs.fau.de

Abstract

Fault and intrusion tolerance is an important paradigm for building distributed systems that work in spite of accidental and malicious faults. This paper discusses how to harness virtualization technology for building such dependable systems. We show that virtualization promotes a hybrid fault model that allows tolerating malicious intrusions in application domains with little overhead. The proposed architecture features mechanisms for supporting heterogeneity of the replicas. A hypervisor-based replication controller achieves perpetual operation through periodic proactive recovery of the replicas. Re-mapping of state storage between virtual machines speeds up the state transfer of a stateful replicated service. Our VM-FIT prototype implements the core functionality of such a virtualization-based replication architecture. We present some performance measurements and close with a discussion of future research directions.

1 Introduction

The ability to operate correctly in spite of the occurrence of accidental and malicious faults is becoming an important requirement of distributed applications. Intrusion tolerance [24] has become popular as a paradigm for building systems that function correctly in spite of intrusions. This paper discusses the use of virtualization technology for the construction of fault and intrusion tolerant systems.

Virtualization is an old technology that was introduced by IBM in the 1960s [12]. Systems such as Xen [4] made this technology popular on standard PC hardware. Virtualization enables the execution of multiple operating system instances simultaneously in isolated environments on a single physical machine. While mostly being used for issues related to resource management, virtualization can also be used for constructing fault-tolerant systems. The aim of this paper is to investigate to what extent modern virtualization technologies, such as the Xen hypervisor, can be used for constructing dependable systems. We identify the following four main issues:

1. Virtualization technology provides isolation between application domains, the hypervisor, and a privileged system domain. This separation allows adopting a hybrid fault model that supports malicious intrusions within application domains (including the operating system and middleware infrastructure) and a crash-stop model in the isolated system domain.

2. Any virtualization technology provides core mechanisms for the creation and destruction of domains. These mechanisms can be harnessed for designing efficient proactive recovery mechanisms. A timely periodic recovery can be triggered by a recovery service in the isolated system domain. Creating a new domain in parallel to the execution of the other applications permits a reduction of the downtime during recovery to a minimum.

3. Virtualization technology simplifies the introduction of diversity, as the replication logic (in a isolated system domain) can have full control over what operating system and service variant to execute in the application domains.

4. Virtualization can also be applied to disk storage managed by the hypervisor. Virtualized persistent state storage can be used for efficiently re-mapping the state from one application domain to another, and thus allows the implementation of low-cost state transfer strategies.

In the following section, we individually discuss these four aspects of virtualization-based fault and intrusion tolerance. Several of them have been implemented as part of our VM-FIT prototype, which we describe and evaluate in Section 3. Section 4 gives a brief overview of related work, and finally Section 5 concludes.

2 Virtualization-based Fault and Intrusion Tolerance

2.1 Hybrid Fault Model

One key mechanism of virtualization infrastructures such as the Xen hypervisor is the provision of separation between *domains*. In the ideal case, the hypervisor and a protected control domain are fully isolated from one or more application domains that execute

*This work was partially supported by the EC through project IST-2004-27513 (CRUTIAL) and the Large-Scale Informatic Systems Laboratory (LaSIGE).

services. In a *hypervisor-based replication architecture*, application replicas are placed in isolated application domains, while the replication logic resides in a separated system domain.

Virtualization-based replication allows using a *hybrid fault model* that assumes Byzantine failures within application domains and crash-stop failures within the hypervisor and the system domain that contains the replication logic. In this model, the replication controller can use majority voting to detect invalid results from replicas. It is possible to tolerate up to f Byzantine faults using a total number of only $n = 2f + 1$ replicas.

This number is less than the requirements of traditional intrusion-tolerant replication systems, which typically need $n = 3f+1$ replicas [8]. A reduction in the number of required replicas strongly simplifies the provision of independent, heterogeneous implementation versions.

Our VM-FIT architecture [18, 19] is a generic infrastructure for the replication of network-based services on the basis of the Xen hypervisor. VM-FIT uses the hypervisor technology to provide communication and replication logic in a privileged domain, while the actual service replicas are executed in isolated guest domains.

In the RESH (Redundant Execution on a Single Host) variant, VM-FIT allows the redundant execution of a service on a single physical host. In this variant, multiple application domains are used to deploy replicas on a single host. This approach allows the toleration of non-benign random faults in the replicas,such as undetected bit errors in memory, as well as the toleration of software faults by using N-version programming. The REMH (Redundant Execution on Multiple Hosts) supports replication on multiple machines using the same core architecture. The main difference to RESH is the integration of group communication facilities in the replication logic. This variant allows tolerating full crashes of some of the replica hosts, instead of tolerating faults only within a virtual domain.

2.2 Proactive Recovery

Traditional intrusion-tolerant systems [7,8,15] typically use Byzantine fault tolerant replication algorithms, which are able to tolerate a finite number of f faults in group of n replicas. However, these systems face the problem that, given sufficient time, an adversary might be able to compromise more than f replicas. Proactive recovery [9, 16] has been proposed as a solution to overcome this limitation of intrusion-tolerant systems. The core idea is to periodically recover all replicas by reinitializing them from a secure base. For example, a tamper-proof external hardware might be used for rebooting the node from a secure code image. This approach removes potential intrusions; as a result, the number of replica failures that the system can tolerate is limited only within a single recovery round, but is unlimited over the system lifetime.

Under the assumption of malicious faults, it is not possible to trigger the recovery within service replicas, as an intruder can inhibit the recovery of the replica. A classic approach is using dedicated hardware that resets and reinitialises a replica periodically. Using virtualization, the replication logic in a separated, intrusion-free domain can be used as a trusted entity that is able to completely re-initialise the replica domains, without requiring dedicated hardware. The recovery operation initializes the complete replica (including operating systems, middleware, and service instance) with at “clean” state, securely obtaining the service state from other replicas with a fault-tolerant state transfer protocol.

Proactive recovery, however, may reduce availability, as the recovery of a replica reduces the number of available replicas (see also Sousa et al. [21]). This disadvantage can be compensated by increasing the number of replicas, but this not only increases hardware costs and run-time costs, but also makes it more difficult to maintain diversity of the replica implementations. A different approach is to try to minimize the time needed for recovery.

Virtualization technology can help building efficient proactive recovery infrastructures [18]. The hypervisor can be used to shut down and restart a replica running in a virtual machine. In addition, the new replica instance can be started using an additional virtual machine in parallel to the execution of the old replica. This allows a substantial reduction of downtime during recovery, as the boot process does not affect replica availability. After initialisation of the new replica, the replication coordinator can shut down the old replica and trigger the activation of the new one. This approach has some impact on service performance, as the local resources (such as CPU and memory) have to be shared between the replica domains. But on the other hand it minimises the time of complete replica unavailability to the time necessary for the coordinated state transition form the old replica instance to the new one.

2.3 Diversity

Diversity of replicas is essential in intrusion tolerant systems in order to avoid that an adversary can exploit the same vulnerability multiple times within a short time interval. A virtualization-based recovery mechanisms provides an ideal basis for introducing diversity both in space and in time.

A hypervisor-based replication architecture allows a *transparent interception* of the client–service interaction, independent of the guest operating system, middleware, and service implementation. As long as the assumption of deterministic behaviour is not violated, the service replicas may be completely heterogeneous, with different operating systems, middleware, and service implementations.

On the one hand, this independence simplifies the use of heterogeneous versions in the application domains. There is no need to integrate the replication logic in each replica variant. Instead, a common replication mechanisms is implemented in the protected domain. The replica implementations still have to provide some prerequisites for replication, such as having deterministic behaviour and supporting transformation of version-specific state representation in a common abstract state format.

On the other hand, hypervisor-based recovery can be used to provide diversity in time. For example, the operating system can be changed in the new version, or internal configurations can be modified on each recovery reboot. Address space randomization is another popular technique for obtaining diversity.

The provision of multiple deterministic versions of complex applications is not a trivial case. In the FOREVER project [1], we plan to further investigate the introduction of diversity with a virtualization-based recovery service.

2.4 State Transfer

In a stateless replication system, the transition from an old to a new replica version is almost instantaneous. In a stateful system, after creating a new instance of operating system, middleware, and service, the new replica needs to be initialized with the service state, which requires a state transfer. The state transfer increases the time that a replica is unavailable during proactive recovery, as request execution has to be stopped during the creation of a state checkpoint. Furthermore, the state transfer also affects service operation by consuming network and CPU resources.

Virtualization technology can be used to enhance the state transfer to the new replica. During a proactive recovery operation, having both old and new replica running in parallel on a single machine enables a simple and fast state copy in the case that the old replica is not faulty; this fact can be verified by taking a distributed checkpoint on the replicas and verifying the validity of the local state using checksums. Only in the case of an invalid state, a more expensive remote state transfer is necessary.

State transfer must also cope with heterogeneity between replicas. Heterogeneity is introduced by diversity. The transformation of state into local replica-specific representations needs some support from replica implementations. The generic state transfer mechanisms must provide means to adapt the state accordingly, by interacting with this application-level adaptation functionality. We are currently investigating how the disk virtualization mechanisms of the Xen hypervisor can be exploited in order to optimize the state transfer on the basis of disk snapshots and disk remapping.

3 VM-FIT Prototype

The VM-FIT prototype [18, 19] supports hypervisor-based replication of network-based services. A replication controller in a privileged domain intercepts client interaction with a replicated service, handles communication with replicas and voting on replies, and provides support for proactive recovery.

3.1 Architectural Overview

The VM-FIT prototype implements the basic system architecture shown in Figure 1. Service replicas are executed in isolated domains (Dom. Guest). The network interaction from client to the service is handled by the replication manager in the system domain (Dom. 0). The manager intercepts the client connection and distributes all requests to the replica group using the Spread group communication system [2]. Each replica processes the client requests and sends a reply to the node that accepted the client connection. At this point, the replication manager selects the correct reply for the client using majority voting.

The replication logic provides mechanisms for instantiating and initialising service replicas. For each replica variant, a disk image of a Xen virtual machine with a preconfigured operating system and middleware environment has to be provided. After domain initialisation, a state transfer from other replicas to the new domain is triggered. In our prototype, we assume that a secure code basis for

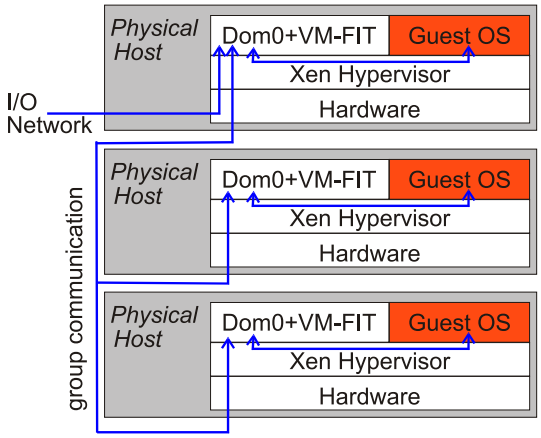


Figure 1. VM-FIT basic replication architecture

the replica is available locally, and only the data state is required to initialise the replica. We assume that the replication logic can request an application-controlled serialisation of the state in an abstract format.

The replication logic also offers support for proactive recovery. Unlike other approaches, the hypervisor-based approach permits the initialisation of the rejuvenated replica instance concurrent to the execution of the old instance. The hypervisor is able to instantiate a second Domain Guest on the same hosts. After initialisation, the replication coordinator can shut down the old replica and trigger the activation of the new one.

The state of the rejuvenated replica needs to be initialised on the basis of a consistent checkpoint of all replicas. As replicas may be subject to Byzantine faults and thus have an invalid state, the state transfer has to be based on an majority agreement of all replicas. The VM-FIT architecture uses the local state of the old replica version on the same host. This state is transferred locally to the new replica, combined with a verification of its validity on the basis of checksums obtained from other replicas. Only if the local state is invalid, a remote state transfer becomes necessary.

The checkpointing and state transfer are time-consuming operations. Furthermore, their duration depends on the state size. During the checkpoint operation, a service is not accessible by clients; otherwise, concurrent state-modifying operations might cause an inconsistent checkpoint. Consequently, there is a trade-off between service availability and safety gained by proactive recovery given by the recovery frequency of replicas. To reduce the unavailability of a service, while still providing the benefits offered by proactive recovery, more than one replica could be recovered at a time. However, in previous systems with dedicated hardware for triggering recovery, the number of replicas recovering in parallel is limited by the fault assumption, as every recovering replica reduces the number of available nodes in a group and, consequently, the number of tolerable faults.

The VM-FIT architecture is able to offer a parallel recovery of all replicas simultaneously. If service replicas have to be recovered, every node receives a checkpoint message and determines the replica state. The replication logic receives this state and prepares a

shadow replica domain. This domain will later be used replace the existing local replica instance and is initialised by the state transfer operation. This approach reduces the downtime due to checkpointing to one checkpoint every recovery period.

3.2 Experimental Results

The current VM-FIT prototype uses the Xen 3.0.3 hypervisor and Linux kernel 2.6.18 both for Domain 0 and for the replica domains. In the following, we describe a few experiments that evaluate the basic proactive recovery scheme, which have first been published in [18]. The experiments examine the behaviour of the VM-FIT proactive recovery architecture for replicating a service on a modern server machine (Sun X4200 server with two dual-core Opteron CPUs at 2.4 GHz and 1 GBit/s switched Ethernet).

In the experiments, a single client on a separate machine sends requests via a LAN network to the service host, which runs 3 replicas of the same network-based service. The replicated service has a very simple functionality: on each client request, it returns a local request counter. It is a simple example of stateful service, which requires a state transfer upon recovery (i.e., the initialization of a new replica with the current counter value). As a performance metric, we measure the number of client requests per second, obtained by counting the number successful requests in 250ms intervals at the client side; in addition we analyse the maximum round-trip time as an indicator for the duration of temporary service unavailability.

We study four different configurations. The first configuration does not use proactive recovery at all. The second configuration implements a "traditional" recovery strategy; every 100s, a replica, selected via a round-robin strategy, is shut down and restarted. A distributed checkpoint of the application state is made before shutting down a replica. This checkpoint ensures that the system can initialize a replica with a correct state (validated by at least $f + 1$ replicas), even if a replica failure occurs concurrent to a recovery operation. The third configuration uses the virtual recovery scheme proposed in this paper: it first creates a new replica instance in a new virtual machine, and then replaces the old instance in the group with the new one. The last configuration uses the same basic idea, but restarts all replicas simultaneously.

The recovery frequency (one recovery each 100s) was selected empirically such that each recovery easily completes within this interval. Typically, a full restart of a replica virtual machine takes less than 50s on the slow machines, and less than 20s on the fast machines. In configuration 4, the recovery of all replicas is started every 300s. This way, the frequency of recoveries per replica remains the same (instead of recovering one out of three replicas every 100s, all replicas are recovered every 300s).

Furthermore, the measurements include the simulation of malicious replicas. Malicious replicas stop sending replies to the VM-FIT replication manager (but continue accepting them on the network), and furthermore perform mathematical computations that cause high CPU load, in order to maximize the potential negative impact on other virtual machines on the same host. Malicious failures occur at time $t_i = 600s + i * 400s, i = 0, 1, 2, \ldots$ at node i mod 3. This implies that the frequency of failures (1/400s) is lower than that of complete recovery cycles (1/300s), consistent with the assumptions we make.

variant \ time	100s.. 400s	650s.. 950s	1050s.. 1350s	600s.. 1800s	max. RTT
A	4547	4479	0	(-)	∞
B	4502	3879	3726	3702	45s
C	4086	4046	4112	4067	1s
D	4169	3992	3960	3992	<250ms

Table 1. Average performance (requests/s) and worst-case RTT observed at the client on a multi-CPU machine

The measurement in Figure 2 shows that, without proactive recovery, there is no significant performance degradation after the first replica fault. Due to the availability of multiple CPUs, each replica can use a different CPU, and thus the faulty replica has (almost) no negative impact on the other replicas. After the second replica failure, the service becomes unavailable. In variant (B), periodic recovery works well in the absence of failures ($t < 600s$). The recovering replica disconnects from the replica group, and thus the replica manager has to forward requests only to the remaining nodes, resulting again in a speed-up during recovery. A faulty node in parallel to a replica recovery, however, causes periods of unavailability (see markers on X-axis). In variant (C), there is no noticeable service degradation during replica recovery. The only visible impact are two short service interruptions, which occur at the beginning of the creation of a new virtual machine and at the moment of state transfer and transition from old to new replica. These interruptions typically show system unavailability during a single $250ms$ measurement interval only. Similar observations also hold for variant (D).

Table 1 shows the average system performance in an interval without failures ($t = 100s \ldots 400s$), after the first failure ($t = 650s \ldots 400s$), after the second failure ($t = 1050s \ldots 1350s$) and in a large interval with failures ($t = 600s \ldots 1800s$). It can be observed that the first recovery strategy (B) has almost no influence on system throughput; variants (C) and (D) reduce the performance of the service by 10% and 8%, respectively, during the period without faults. With faulty replicas, the average throughput drops significantly in variant (B) due to the temporary service unavailability, while it remains almost constant in the case of (C) and (D).

3.3 Discussion

The measurements demonstrate that the VM-FIT proactive recovery schemes (C and D) are superior to the simple one (B). While there is not much difference in the average throughput, the simple scheme causes long periods of unavailability, which is undesirable in practice. The unavailability could be compensated by increasing the number of replicas. In practice, this would make implementation diversity more difficult (more different versions are needed). Furthermore, in a virtual replication scenario on a single physical host, adding another replica on that host would reduce the system performance.

The experiments only considered replication a single physical machine. The same proactive recovery mechanisms can also be used in VM-FIT for replication on multiple physical hosts. In this case,

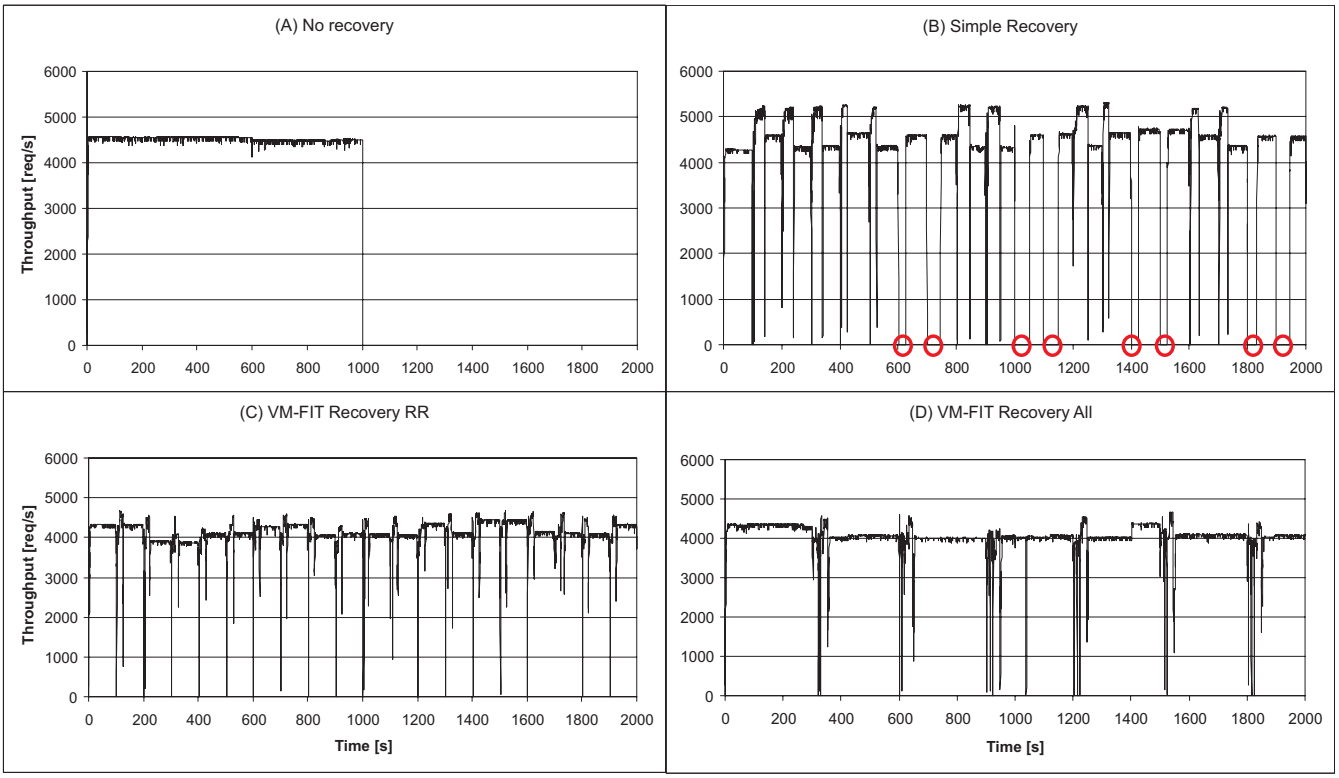


Figure 2. Throughput measurements on multi-CPU machine

client requests are distributed to all nodes using totally ordered group communication. The request distribution is the same for all variants of proactive recovery and thus will not have much impact on the relative performance. The main difference will be that there is no impact of a recovering node on the other replicas.

In the prototype, no attempts towards formal verification of the trusted component have been made. Indeed, the same operating system, an off-the-shelf Linux distribution, is used for Domain 0 and Domain Guest. As a consequence, vulnerabilities at the operating system level are currently present in both domains. We expect, however, that this is only a limitation of the early prototype. In future work we plan to use a hardened Linux system as Domain 0, or even use a minimalistic operating system that might permit formal verification.

4 Related Work

Virtualization has become popular on standard PC hardware by systems such as Xen [4] and VMware [22]. Virtualization enables the execution of multiple operating system instances simultaneously in isolated environments on a single physical machine. The L4Ka microkernel also offers virtualization functionality [13]. In addition, significant efforts towards a formal verification of the L4 kernel have been made by other researchers [23]. These results provide an excellent basis for justifying a hybrid fault model that assumes an intrusion-free hypervisor and system domain.

While mostly being used for issues related to resource management, virtualization has previously been applied for constructing fault-tolerant systems. Bressoud and Schneider [5] demonstrated the use of virtualization for lock-stepped replication of an application on multiple hosts.

Besides such direct replication support, virtualization can also help to encapsulate and avoid faults. The separation of system components in isolated virtual machines reduces the impact of faulty components on the remaining system [14]. Furthermore, the separation simplifies formal verification of components [23]. In this paper, we do not focus on these matters in detail. However, such solutions provide important mechanisms that help to further justify the assumptions that we make on the isolation and correctness of a trusted entity.

Using virtualization is also popular for intrusion detection and analysis. Several systems transparently inspect a guest operating system from the hypervisor level [10, 11]. Such approaches are not within the scope of this paper, but they are ideally suited to complement our approach. Intrusion detection and analysis can be used to detect and analyse potential intrusions, and thus help to pinpoint and eliminate flaws in systems that could be exploited by attackers.

Several authors have previously used proactive recovery in Byzantine fault tolerant systems [3, 6, 9, 16]. It is a technique that periodically refreshes nodes in order to remove potential intrusions. The BFT protocol of Castro and Liskov [9] periodically creates stable checkpoints. The authors recognize that the efficiency of state

transfer is essential in proactive recovery systems; they propose a solution that creates a hierarchical partition of the state in order to minimize the amount of data to transfer.

Sousa et al. [21] specifically discuss the problem of reduced system availability during proactive recovery of replicas. The authors define requirements on the number of replicas that avoid potential periods of unavailability given maximum numbers of simultaneously faulty and recovering replicas. Our approach instead reduces the unavailability problem during recovery by performing most of the initialization of a recovering replica in parallel to normal system operation using an additional virtual machine.

In a recent publication, Silva et al. [20] use an approach similar to ours for software rejuvenation. The main difference is that these authors focus on recovering from error situations caused by "software ageing".

Ramasamy and Schunter [17] use combinatorial modelling to analyse how the use of virtualization can affect system dependability. Such a careful analysis allows a better judgement on the conditions that are necessary to make a system such as VM-FIT more reliable than non-replicated one.

5 Summary

This paper discussed the benefits that virtualization offers for constructing fault-tolerant systems. The most important benefits of such an approach are the use of a hybrid fault model that allows tolerating malicious intrusions with a minimum number of replicas; the support for heterogeneous replicated applications, middleware and operating systems on top of a hypervisor-based replication infrastructure; the support for efficient proactive recovery operations; and the potential for optimized checkpointing and state transfer using virtualization mechanism.

References

[1] FOREVER: Fault/intrusiOn REmoVal through Evolution & Recovery; http://forever.di.fc.ul.pt/.

[2] Y. Amir, C. Nita-Rotaru, J. Stanton, and G. Tsudik. Secure spread: An integrated architecture for secure group communication. *IEEE Trans. on Dependable and Secure Computing*, 02(3):248–261, 2005.

[3] B. Barak, A. Herzberg, D. Naor, and E. Shai. The proactive security toolkit and applications. In *CCS '99: Proc. of the 6th ACM conference on Computer and communications security*, pages 18–27, New York, NY, USA, 1999. ACM Press.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proc. of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

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

[6] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *CCS '02: Proc. of the 9th ACM conference on Computer and communications security*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[7] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the internet. In *Intl. Conf. on Dependable Systems and Networks*, pages 167–176, 2002.

[8] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI '99: Proc. of the 3rd Symp. on Operating Systems Design and Implementation*, pages 173–186. USENIX Association, 1999.

[9] M. Castro and B. Liskov. Proactive recovery in a byzantine-fault-tolerant system. In *4th Symp. on Operating Systems Design and Implementation (OSDI)*, San Diego, USA, Oct. 2000.

[10] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.*, 36(SI):211–224, 2002.

[11] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.

[12] R. P. Goldberg. Architecture of virtual machines. In *Proc. of the workshop on virtual computer systems*, pages 74–112, New York, NY, USA, 1973. ACM Press.

[13] J. LeVasseur, V. Uhlig, M. Chapman, P. Chubb, B. Leslie, and G. Heiser. Pre-virtualization: soft layering for virtual machines. Technical Report 2006-15, Fakultät für Informatik, Universität Karlsruhe (TH), July 2006.

[14] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proc. of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004.

[15] D. Malkhi and M. Reiter. Byzantine quorum systems. In *STOC '97: Proc. of the twenty-ninth annual ACM symposium on Theory of computing*, pages 569–578, New York, NY, USA, 1997. ACM Press.

[16] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *PODC '91: Proc. of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59, New York, NY, USA, 1991. ACM Press.

[17] H. V. Ramasamy and M. Schunter. Architecting dependable systems using virtualization. In *Workshop on Architecting Dependable Systems: Supplemental Volume of the 2007 International Conference on Dependable Systems and Networks (DSN-2007)*.

[18] H. P. Reiser and R. Kapitza. Hypervisor-based efficient proactive recovery. In *Proc. of the 26th IEEE Symposium on Reliable Distributed Systems - SRDS'07 (Oct 10-12, 2007, Beijing, China)*, pages 83–92, 2007.

[19] H. P. Reiser and R. Kapitza. VM-FIT: supporting intrusion tolerance with virtualisation technology. In *Proceedings of the 1st Workshop on Recent Advances on Intrusion-Tolerant Systems (in conjunction with Eurosys 2007, Lisbon, Portugal, March 23, 2007)*, pages 18–22, 2007.

[20] L. M. Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak. Using virtualization to improve software rejuvenation. In *Proc. of the 6th IEEE Int. Symp. on Network Computing and Applications (NCA 2007)*, volume 00, pages 33–44. IEEE Computer Society, 2007.

[21] P. Sousa, N. F. Neves, P. Verissimo, and W. H. Sanders. Proactive resilience revisited: The delicate balance between resisting intrusions and remaining available. In *SRDS '06: Proc. of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, pages 71–82, Washington, DC, USA, 2006. IEEE Computer Society.

[22] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001.

[23] H. Tuch, G. Klein, and G. Heiser. Os verification — now! In M. Seltzer, editor, *Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS X)*, 2005.

[24] P. E. Veríssimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems*, volume Volume 2677/2003, pages 3–36. Springer Berlin / Heidelberg, 2003.

Server Virtualization - basic building block for Dynamic IT

Karsten Beins
Fujitsu-Siemens Computers GmbH
Karsten.beins@fujitsu-siemens.com

Abstract

Virtualization on storage, network, server, application and desktop layers can establish abstractions which are highly beneficial for IT infrastructure operation. This paper focuses on server virtualization options, examining their maturity and remaining challenges. The FlexFrame Infrastructure approach of Fujitsu Siemens Computers is shown as an example how to address the management complexity of server virtualization. For Fujitsu Siemens Computers server virtualization is a key technology to enable an innovative transition towards a Dynamic Data Center.

1. Introduction

At a high level, virtualization is an intermediary hardware or software layer that enables separation of the logical view from the physical view to resources, such as CPU, memory, disk, network, I/O connectivity. Without virtualization every server, operating system or application have to manage their dedicated physical resources.

In a virtualized environment, the server, operating system or application do not need to know the exact physical implementation or version of a resource and where the resource resides physically. Logical resources that are presented in a virtualization layer replace server's, operating system's or application's physical resources. The virtualization layer then dynamically maps these logical resources to the target physical resources.

Virtualization enables greater flexibility and efficiency in resource assignment to a server, operating system or application. Virtualization also allows the IT department to separate deployment, life cycle management and innovation decisions regarding server, operating systems and application from those regarding compute and storage devices. Virtualization technology enables a paradigm-shift towards dynamic and automated data centre operation.

2. Layers of Virtualization

Various virtualization techniques exist. They all make their individual contributions to establish abstractions on different layers in an IT infrastructure. Applying a virtualization taxonomy (inspired by the storage virtualization taxonomy definition of the SNIA[2]) helps to differentiate those techniques in a systematic way based on three orthogonal aspects (see figure 1):

- *What is created?*
- *Where is it done?*
- *How is it implemented?*

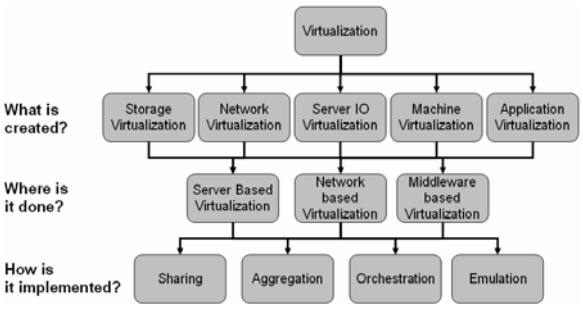


Figure 1. Virtualization Taxonomy

Below we differentiate major virtualization techniques at different layers (see figure 2) depending on the functionality they create and characterize their "where" and "how". For the rest of this paper we will focus on the first three server virtualization-related approaches.

(1) *Application Virtualization*:
At the application layer, preparations are made to flexibly run application programs on different instances of OS and servers, by programming techniques which avoid tight binding to the underlying OS and server hardware. This is either accomplished *directly* by individual application programming. Examples are the SAP business suite and Oracle 10g database and application servers.

Alternatively, *application containers* provide the runtime environment for applications. For those containers a clean separation from the underlying OS and server hardware is implemented, thereby inheriting the desired capability to the contained applications. Examples are applications running on Java VMs, in Solaris zones and in Linux or BSD jails.

In both cases multiple applications can run on the same server, sharing its resources and the same OS instance (that is, all applications must use the same OS type and patch level). The container-based approach provides a higher level of isolation between different applications, by cleanly separating the user mode execution contexts including important system-wide name spaces, such as the (root) file system. In addition some container virtualizations provide resource management per container. As a result multiple applications can not directly affect each other's execution by accident or maliciously. However, instability or resource issues at the kernel mode execution affect all applications.

(2) *Virtual Machines:*

At the machine layer, special system software emulates a server's complete hardware / software interface, thereby creating *Virtual Machines* (aka *Virtual Servers*), which can be used (almost) like the real servers with the same hardware / software interface.

One or more Virtual Machines can run simultaneously on the same physical computer by sharing its real resources. As a key difference to application virtualization, in every Virtual Machine its own OS instance (guest OS) is running,. This allows heterogeneous OS deployments on the same real server and extends the isolation between different applications to include the kernel mode execution level, at the price of higher execution efforts compared to application virtualization.

Two major flavors of the emulation system software can be distinguished: a) host OS-based, as special application on top of a host OS, b) Hypervisor-based, directly controlling the underlying real computer hardware instead of an OS.

(3) *Server Virtual I/O Connectivity* (LAN, SAN)

At the physical server layer abstractions for I/O connectivity into Ethernet LANs and Fibre Channel SANs are introduced to accomplish goals like a) cable / switch consolidation and b) transparent replacement / movement of a server regarding its I/O identities (e.g. MAC addresses, WWPN) to avoid expensive configuration changes in the LAN and SAN switches. Such Virtual I/O for physical servers must be seen as orthogonal to the virtualized I/O implemented by Virtual Machines.

(4) *Storage Virtualization*

At the storage layer many techniques exist to provide OSes and applications the required access services to storage objects like LUNs or files with well defined service levels, but keeping implementation aspects like RAID levels, multi pathing, replication, backup, etc. transparent.

(5) *Network Virtualization*

At the network layer many techniques exist to provide OSes and applications running on servers the required connectivity services to other servers, storage, clients, etc. with well defined service levels, but keeping implementation aspects transparent, such as routing, VLAN / VSAN isolation, quality of service, etc..

(6) *Desktop Virtualization*

At the layer of clients a trend can be seen to use virtualization techniques to replace traditional PC installations by thinner client forms without a local desktop installation. Instead, desktop software is run in servers in the data center ("back-racking"), communicating to the human interface components of the client over the network (with protocols like RDP etc.)

For desktop software execution in the data centre all kinds server virtualization (see above) are available. This may substantially reduce the efforts of central desktop management in many ways, e.g. desktop software deployment and patching, backup / restore and security.

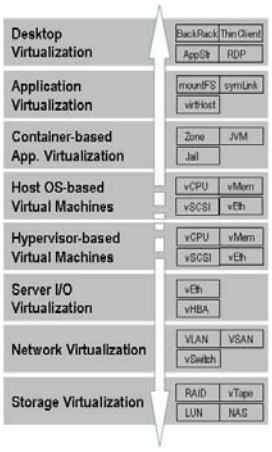


Figure 2. Virtualization Layers

3. Business benefits from Virtualization

Virtualization does not represent a business value by itself, however, customers are highly interested to turn delivered advanced technical capabilities like abstraction, isolation, resource sharing, live migration, etc. into business benefits like cost optimizations (CAPEX, OPEX, TCO), higher Service Levels, increased agility and greater flexibility. Main targets of Server Virtualization approaches are:

- *Server Consolidation*:
  Running multiple applications safely on the same physical server can shift average server utilization from very low to highly efficient and substantially reduce the number of required servers with resulting CAPEX and OPEX benefits. Technology candidates are Application Virtualization, Application Containers and Virtual Machines.
- *Hardware / Software Separation*:
  Virtual Machines create a homogeneous hardware / software interface for virtual servers that can be kept stable over a long period. By using VMs, software solution stacks (application, middleware, OS) can be managed separate from the underlying hardware, enabling customers to handle innovation decisions and life cycle management for hardware and software independent of each other and on independent schedules. Customers appreciate new levels of flexibility to reduce their hardware/ software support matrix and to establish cleanly separated organizational responsibilities for HW and SW components in their data centre.
- *Rapid Service Deployment*:
  Customers face the increasing challenge of deploying new services more quickly, to deal with new market demands / opportunities and organizational changes. The overall time to deploy an additional physical x86 server is typically 2-3 months; using VMs in virtualized environments can drive this delay down to minutes. Technical options are Application Containers and Virtual Machines. An additional advantage of Virtual Machines is the option to install pre-tested Application / OS stacks as Virtual Appliances. Thus Server Virtualization can substantially improve business agility.
- *Higher Service Levels (Availability, Performance)*:
  Unlike physical machines Hypervisor-based Virtual Machines can be live-migrated to another physical server without interrupting the service running in that VM. Live Migration can be leveraged to evacuate unhealthy server hardware for proactive maintenance before a crash occurs, and also to react on changed resource demands by moving that service to a server with more or less compute power. Thus Server Virtualization enables higher Service Levels in a cost-effective way.
- *Disaster Recovery*:
  To prepare for disaster many customers establish remote disaster recovery sites, usually including larger farms of servers and storage. In the past the configuration of a disaster recovery site had to be an exact replication of the primary site, which is difficult to create and even more difficult to maintain over time. As a result primary and disaster recovery sites typically exist in a 1:1 relationship. For virtual server farms the identical physical configuration requirement goes away. The necessary virtual servers can be created on a smaller server farm, which can also act as a shared disaster recovery site for multiple primary sites. Thus Server Virtualization can dramatically reduce the CAPEX and OPEX of a disaster recovery site.

Virtualization technology selection decisions depend on customer requirements like performance, application isolation, OS version variety, etc. While it makes sense to use the above virtualization forms stand-alone, many of them can also be combined. Best solutions results are often accomplished by combination.

4. Challenges coming with Virtualization

Many server-related core virtualization technologies are mature enough by now. Multiple commercial and open source products exist, in particular for container-based application virtualization and Hypervisors for Virtual Machines. A majority of large and mid-size companies have already made their own practical experiences and are about ready for broader use even for mission critical workloads.

A complete maturity assessment requires looking also at remaining issues around server virtualization: While server virtualization enables many benefits of more dynamic and automated data centre operation, it introduces new availability and complexity challenges at the same time:

- In the future, hundreds of physical servers will turn into thousands of virtual machines, running on a

smaller number of physical servers. Therefore High Availability and Disaster Recovery become more mandatory because many applications become vulnerable to a single server hardware failure. Traditional High Availability and Disaster Recovery solutions exist but typically cause much higher complexity and resulting CAPEX and OPEX.

- Server Virtualization allows reduction of the physical server count, suggesting a proportional OPEX reduction. However, new tasks and requirements increase the complexity and may eat up the advantage of reduced server count:
  - VMs must be managed in addition to physical servers, creating yet another management domain,
  - VMs must migrate across physical servers without compromising security,
  - VMs must be backed-up in a more efficient way to meet backup time windows.

Moreover, the introduction of Virtual Machine Managers and Hypervisors is currently not satisfying the customer's hope to reduce the complexity of server-OS-application support matrixes. Instead, customers are confronted with a couple difficult selection problems. Hypervisor offerings with different strengths exist: ESX server from market leader *VMware*, various commercialized derivates of *Xen*, and Microsoft *HyperV* is announced for February 2008. While experts expect a rapid functional, performance and robustness convergence and subsequent commoditization of Hypervisors, none of those vendors currently provides support coverage for all relevant OS / ISV application combinations. Therefore customers are forced to either use multiple virtualization products concurrently in their data centre or refrain from virtualized operation of certain applications.

5. FlexFrame Infrastructure

Fujitsu Siemens Computers acknowledges and addresses the above issues and customer pain points with the "*FlexFrame Infrastructure*" platform. The FlexFrame Infrastructure platform consists of a pre-integrated product suite of x86 industry standard servers and virtualization technologies for server I/O and VMs under unified resource management software, and integration and operation service offerings. This platform is a very flexible infrastructure foundation for key business applications in a service-oriented architecture. The platform pre-integration promotes an industrial development of such solutions.

With its common model-driven resource management approach FlexFrame Infrastructure covers

- physical and virtual servers,
- multiple Hypervisors,
- physical server I/O virtualization,
- dynamic orchestration of physical and virtual servers from resource pools,
- automated server high availability
  - incl. server LAN / SAN connectivity
  - based on N+1 or N+N redundancy
  - transparent to OS and applications
- disaster recovery.

Servers are modeled (see figure 3) as aggregates of (physical or virtual) processing resources, virtual SCSI disk and tape resources, and internal and external network connections, which in turn are modeled by virtual Ethernet NICs connected to virtual Ethernet switches.

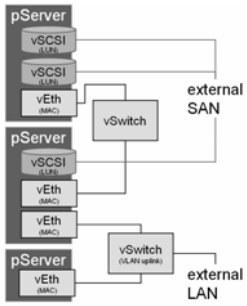


Figure 3. Server Model

Such server configurations are stored as XML-descriptions in a repository. Dynamic server orchestration can then be triggered by GUI or API on demand. Right at that point in time the required resources are allocated from *Processing Area Network* (PAN) resource pools (see figure 4). A PAN can be sub-structured into *Logical PANs* (LPAN), to deal with resource and administrative authority separation, needed for example to server multiple end-customers in the same data centre out of one global resource pool.

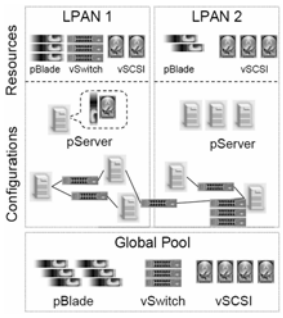


Figure 4. Processing Area Network

5. Virtualization in context of Fujitsu Siemens Computers Dynamic Data Center architecture

Fujitsu Siemens Computers look at Virtualization in a broader context within the data centre infrastructure. The Dynamic Data Center (DDC) [1] architecture (see figure 5) from Fujitsu Siemens Computers describes an IT architecture that targets a breakthrough compared to existing paradigms. The focus of the DDC is on services that are provided to end users and the Service Level Agreements (*SLAs*) that define the quality of provided services in terms of response times, availability, etc. The demanded quality should be offered at the lowest possible price (*efficiency*), and the IT in question should be able to adapt rapidly to changes in business processes (*agility*).

The DDC is therefore based on new hardware and software architectures that enable greater efficiency and agility, yet the appropriate high reliability. These architectures are complemented by concepts for integration into existing IT operations.

In dynamic IT infrastructures ideally every service can run on any system and be relocated between systems in a short time.

A first key step towards that vision is to break up fixed bindings between services and dedicated hardware. All hardware resources (e.g. storage, servers or server components) are grouped in pools, from which they can be dynamically allocated, orchestrated and then assigned to services on demand. Resources can be repurposed over time as required by changing resource demand.

This leads to two desired transitions: replace traditional silo-like application architectures by dynamic and service oriented architectures, and make configuration sizing to peak load unnecessary.

Virtualization plays an enabling role in this paradigm change: it creates the necessary abstraction between application software stacks and resources. Servers and storage can be dynamically orchestrated from resource pools but still behave like traditional computers from the applications point of view, as expected.

In the next step operations management of virtual and real resources is largely automated. Resources are automatically provided to services on the basis of predefined policies. For example, if resources fail or are in short supply, corrective actions can be initiated full- or semi-automatically. Services support business processes and come along with defined SLAs. Goal of the policies is to ensure that SLAs can be kept automatically.

Virtualization and Automation are complemented by Accounting and Billing to charge the services according to their dynamic resource usage.

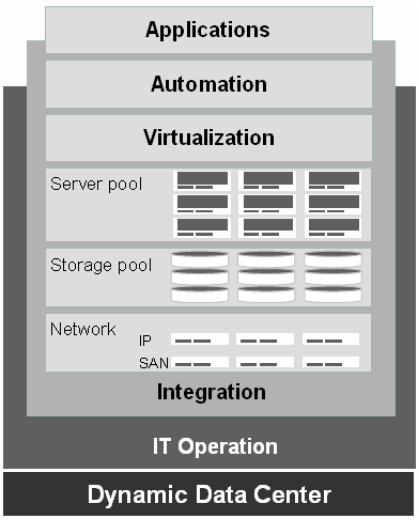


Figure 5. Dynamic Data Center Architecture

10. References

[1] Fujitsu Siemens Computers, "The Architecture for Flexible Enterprise IT Dynamic Data Center", http://www.fujitsu-siemens.com/it_trends/dynamic_data_center/info/index.html, Munich, 2007

[2]Bunn, F., Simpson, N., Peglar, R., Nagle, G., "Storage Virtualization", SNIA Technical Tutorial, 2003

Implementierungen

Virtualizing an IT Lab for Higher Education Teaching

Nils gentschen Felde, Tobias Lindinger
Munich Network Management Team
Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 Munich, Germany
{felde|lindinge}@mnm-team.org

Helmut Reiser
Munich Network Management Team
Leibniz Computing Centre
Boltzmannstr. 1, 85748 Garching, Germany
reiser@mnm-team.org

Abstract

In universities, a great amount of time is needed to manage and operate lab course IT infrastructures. Additionally, university's resources are occupied and teaching staff is needed to supervise the attending students.

In this paper, we present a concept for planning and deploying virtualized IT infrastructures (hosts and network) for higher education purposes and show an implementation including tool supported management of the virtual environment. The management platform facilitates the administration of virtual machines by students and thus frees the teaching staff from that duty. As a proof of concept, a number of different teaching environments used in a lab course on IT security have been virtualized. The course is intended for graduate students and poses high demands on the infrastructure, its availability and its performance, while security aspects have to be taken into account. Concluding the paper, experiences made during two years of productive use as well as updating the system to new releases of the virtualization software are pointed out.

1 Introduction

The Ludwig-Maximilians-Universität München and the Technische Universität München offer a practical course on IT security for graduate students. In this context, multiple workstations and servers are provided. Over time, defects of hardware components occur more often, which demand human interaction in order to ensure further operation of the lab. Moreover, the infrastructure is only accessible during certain days of the week and for a limited amount of time due to the institute's opening hours. As the course is attended by students of two different universities located at different places, the students' time of travel is considerably high as well. In order to improve the situation and save valuable time of the teaching staff, the virtualization of the whole lab course seems a suitable solution.

1.1 The Lab Course Use Case

The IT security lab course mainly deals with configuration aspects of network components and IT services. Security flaws are explained and the misuse of those illustrated in experimentals using sniffers, portscanners, several hacking tools and executing Denial of Service (DoS) attacks. Securing networks, their components and IT services are tasks students have to deal within the course. The course has a maximum capacity of 40 students working together in groups of two, each group having two computers at hand. During the course, several different network topologies are needed. Thus, a mechanism for simple and dynamic adaption of the infrastructure is necessary.

1.2 Requirements

In the context of the practical lab course, four major requirements have to be fulfilled while designing and implementing the lab course infrastructure:

1. Security.
   Due to the fact that the course deals with IT security, one important factor while designing the virtual lab is defined by IT security itself. Security aspects of the underlying host system are a primary issue in order to guarantee a highly available and secure course environment. As some critical experiments like DoS attacks and password cracking are carried out within the course, the protection of the outer world is an important fact as well, while access to the Internet is necessary to download software components.

2. Transparency.
   The virtualization must not be visible to the students. No student needs to have any access to or knowledge of the underlying physical hardware components. Students don't even have to know about the virtualization in order to work with the components provided.

3. Accessibility.
   Access to the machines should be possible from any workstation connected to the Internet, including both console access and the use of graphical user environments in a secure manner with adequate performance supporting small bandwidth Internet connections as for example ISDN or even analog dial-up connections. Besides, a large variety of operating systems used by the students has to be supported in order to connect to the lab. In particular, a minimum of Apple MAC OS, Microsoft Windows and Linux/UNIX on the client side should be usable, while the virtual machines themselves are based on Linux without exception.

4. Management.
   Management aspects have to be separated into two major dimensions:

   (a) Management of the virtual lab infrastructure.
   To ease the management of the virtual lab is a major requirement, meaning that it has to be comparatively easy to keep the lab up and running and to ensure a secure environment for the experiments. This discipline is left to the teaching staff and system administrators, as it only deals with the hosting system itself and not with the virtual workstations.

   (b) Management of the virtual machines.
   The management of the virtual workstations shall be left to the students, releasing the teaching staff and system administrators from that duty. It has to be possible for all the students participating in the course to manage their own virtual machines in a comfortable way. In particular, they have to be able to restart their machines if a problem occurs, create snapshots as backups or even reinstall a clean system image in case of a major misconfiguration. These operations should not be allowed to be executed on foreign virtual machines related to other students.

1.3 Contribution of this Paper

This paper describes how to migrate an existing lab infrastructure to a virtual lab infrastructure taking security, transparency, accessibility and management aspects into account. The implementation shown in section 4 contains more than 40 virtual machines (also referred to as *VM*s) including both workstations and servers, all of them having multiple network interface cards, more than 20 virtualized bridges, hubs or switches hosted by only one physical machine. Additionally, different network topologies are implemented, having the opportunity to switch between them dynamically using prebuilt scripts. The virtual machines are accessible using graphical desktop environments or secure shells 24 hours a day. The usage of virtual private network (VPN) technologies completes the implementation.

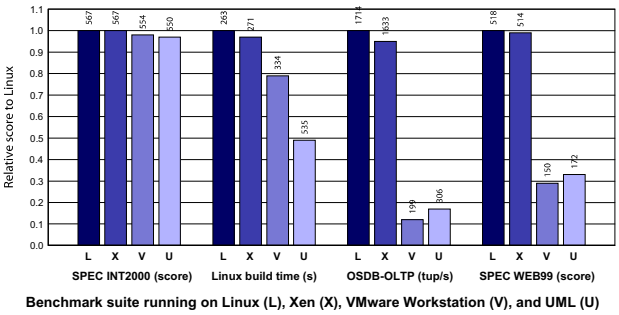


Figure 1. Virtualization benchmarks [7]

The remainder of the paper is structured as follows. First, Xen is introduced in section 2 as it will be the virtualization tool of choice for the implementation later on. Following, the concept, implementation and deployment of the virtual lab course is described and the fulfillment of the before mentioned requirements is shown. Concluding the paper, a short overview of the performance of the implementation experienced in real life usage is given in section 5 and some possible improvements and further work is pointed out in section 6.

2 State of the Art and Related Work

Beside virtual machines, network components like switches, hubs and firewalls as well as their connections have to be virtualized. These facts raise some additional requirements for the implementation of the virtual course infrastructure. Extensive tests [4] which virtualization technique is suitable for our usecase have been carried out and resulted in using Xen.

The next section introduces Xen as an example for host virtualization as it was the fastest platform (see figure 1) available when we started the project three years ago in 2005. Additionally, the network setups can be realized using the techniques and components provided by Xen, whereas User Mode Linux and VMware were too slow or not able to create virtual instances of our network setups due to the lack of several virtual components, in particular hubs. Details related to the implementation can be seen in section 4. Section 2.2 points out related work in the area of virtual lab courses and concludes this section.

2.1 Xen

Xen [12] is a hypervisor that uses the *para-virtualization* concept. Xen provides an interface which is very similar to

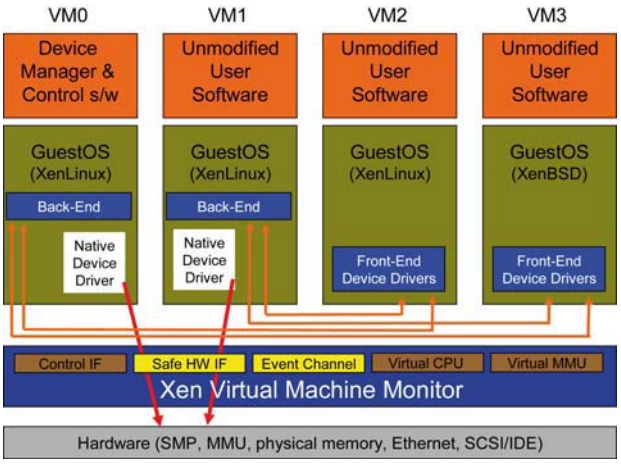


Figure 2. The Xen architecture [8]

the x86 architecture. In order to operate a guest system using Xen, some lines of code of the hosted operating system have to be adapted in order to run using the Xen interface instead of the underlying hardware (e.g. an x86 architecture). Therefore, Xen is only used in combination with open source operating systems as guests (in para-virtualization mode) or technologies like Intel VT-x and AMD-V formally known as Vanderpool and Pacifica.

Figure 2 depicts the layered model Xen implements. The *Xen Virtual Machine Monitor* (VMM, also called the *hypervisor*) [9] introduces an additional layer on top of the hardware of the host system. Via the *Safe Hardware Interface* access to the hardware is granted. The VMM is responsible for every component of the hosting system being accessed by just one system at a time. For example, in case of a CPU (multiprocessor systems are supported) every virtual machine is bound to a *Virtual CPU*. If the virtual machine becomes active that virtual processor is bound to a physical CPU core by the VMM and provides the compute power demanded by the VM.

On top of the VMM all the virtual machines – also called domains – are executed. All domains are treated equally, except the so-called *domain0*. This machine is privileged and its job is to control and manage all the others (the so-called *domU*s). Usually, but not necessarily, domain0 owns access to all physically available hardware components via the hypervisor. This is the reason why there are two different versions of kernels for Xen Linux: One kernel including drivers for the access to the physical hardware that is appointed in domain0 and another kernel without this functionality operated by the guest machines. Both versions can be configured and recompiled manually to add additional features. Frontend drivers for the access to virtual hardware served by the Xen backend system should be included in both versions.

To protect the system from illegal access, Xen makes use of the ring concept of the x86 architecture. Rings – there are four of them, but mostly only two of them are used – represent different access layers. Ring zero represents the kernel mode and ring three is known as the user mode. Xen modifies this mapping as follows: The hypervisor operates in ring zero and the operating system is shifted to ring one. Consequently, the operating systems can be controlled by the hypervisor. OS instances running in ring one are not allowed to execute any privileged instruction on the processor. This is why the operating system has been modified and runs some new functions called *hypercalls* instead of prohibited systemcalls. Trying to pass a systemcall anyhow results in an exception thrown by the processor and is handled by the hypervisor. This only holds true on 32 bit systems, 64 bit systems behave differently.

2.2 Related Projects

Research in the area of virtualizing IT environments used for educational purposes has already been carried out by other groups of researchers. Mostly, the work focuses on the simplification in creating lab infrastructures by booting a number of virtual machines and connecting them to special networks automatically. Usually, this is done according to configuration files built by administrators in advance. Examples include MLN (My Linux Network) [2], VNL (Virtual Networking Lab) [6] and VNUML (Virtual Network User Mode Linux) [11].

The tools developed in these projects ease the process of deploying virtual infrastructures. They also provide tool support for this task, but they are lacking a concept of how to transfer existing lab course infrastructures into virtual environments conveniently. Reconfiguration issues based on easy to use configuration files for whole network setups, as well as per user management interfaces for comfortable and secure remote access to the virtual machines are out of scope.

3 Basic Ideas & Concepts

The fulfillment of the requirements on the lab course infrastructure leads to some obvious attempts. This section presents some ideas on how to conform to these requirements. Afterwards, an implementation of the concept derived in this section is found in section 4.

1. Security.
   To protect the host from attacks originated in VMs, it is necessary to strictly separate the physical system from the VMs. Therefore, the only point of access to the virtual environment is delegated to a VM (the so-called

login server) directly bound to a physical network interface connected to the Internet. In our case, Xen offers a feature allowing the assignment of a physical NIC to a VM exclusively. This feature is granted by the hypervisor.

Using firewalls to prevent unauthorized access to the virtual networks or the management system is a further step towards securing the platform. As communication between the virtual machines via the pre-configured management network (see below) is unwanted, it is prevented by firewall rules. Firewalls implemented in the login server protect the login server from outer world attacks. Also, connections to the Internet can be filtered to prevent attacks from the lab harming foreign resources located outside the lab.

2. Transparency.
   Transparency is guaranteed by virtualizing every single component used for the course environment. Every workstation and every server (see figure 3) is virtualized. Thus, nobody has access to or knowledge of the underlying hardware which serves the infrastructure. This transparency adds additional security to the system. If one of the components is compromised successfully, only one virtual component could be intruded instead of the physical host. Besides, the host itself can be secured by several security means and be placed in a private network segment.

3. Accessibility.
   In order to access the virtual network, a dedicated virtual login server (see figure 3) is used. One of its virtual network interfaces is directly connected to the Internet, while another one connects to a management network. It is either possible to tunnel any traffic through the login server to the designated port on the target machine (e.g. port 22 for SSH) or to connect to the network using VPN technologies. In the latter case, the login server acts as the security gateway and the computer connecting to the VPN becomes part of the management network and thus can access any virtual machine.

4. Management.

   (a) Scripts for booting the scenarios.
   Scripts to start and stop virtualized scenarios are used. The scripts include virtual machine configurations, the creation of network resources e.g. hubs, switches and bridges and the correct wiring of the components. In our case, the creation of these scripts can be simplified using a feature provided by Xen: Parameters can be given and calculations can be performed in the configuration file, which enables the administrators to create virtual machines in a loop within a script. Individual configuration settings of the virtual machines can be calculated in the configuration file depending on the loop parameter.

   (b) Management platform for student use.
   A management platform is introduced in order to enable the participants of the course to manage their own virtual machines. The management includes rebooting, shutting down, backing up, recovering old snapshots and resetting a virtual machine to its initial state as a minimum subset of features. A management interface operated by the hosting system is mandatory for these tasks. A management proxy in the context of the login server grants remote access to the management interface. Making use of reliable authentication and authorization capabilities combined with encrypted data transfer ensures a secure operation of the management platform.

Figure 3 illustrates the basic ideas of the concept this work is based on. A main interest is to isolate the hosting system from the virtual infrastructure due to security aspects.

To ensure the accessibility of the virtual machines, a dedicated management network has to be set up, complementing the teaching network. This enables the users to connect to their virtual machines, regardless any misconfiguration of the interface cards connecting to the teaching network. In order to access the management network, the login server has to be used. A firewall running on this server allows remote access to the VMs, e.g. using SSH tunneling or VPN technologies. Remote logins on the gateway are not permitted for security reasons, of course. Additionally, outgoing traffic to the Internet can be masqueraded using NAT router capabilities, providing Internet access for the teaching network. Connections initiated by virtual machines to the Internet and communication among virtual machines using the management network is not desired and thus not permitted by restrictive firewall rule-sets.

Privileged access to the hosting system is needed in order to manage the virtual machines. For this reason, a management interface is introduced running on the host, which is able to control the VMs (e.g. (re-)booting, shutting down, backing up, recovering old snapshots, ...). To reduce management interactions performed by the teaching staff, a management proxy granting access to the management interface is deployed on the login server. This proxy passes connections originating from the Internet to the management tool transparently. This conserves transparency and enhances security aspects, as using a direct manage-

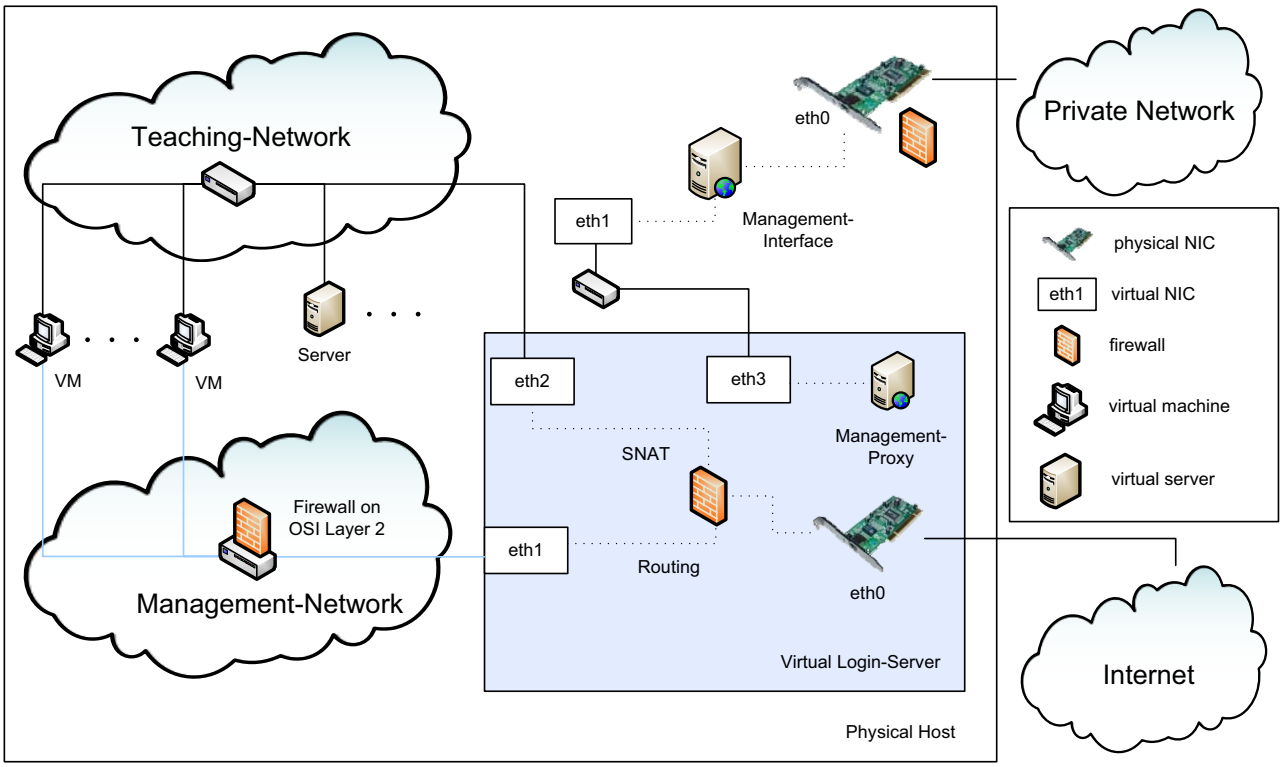


Figure 3. Conceptual view

ment connection to the host would result in granting students access to the host via HTTP and publishing the host's IP address. Of course, additional authentication and authorization processes have to be established. Therefore, a customized authentication handler on the web server that compares the password given to the root password located in the /etc/shadow file in a virtual machine is used. For this purpose the image of a VM is mounted read-only by the web server.

4 Deployment

Figure 4 illustrates the instantiation of the concept presented above for a lab course provided by the two universities. In this section, first the basis for the implementation is described briefly, including the hardware of the hosting server as well as the software chosen for the virtualization process. Section 4.3 gives a detailed overview of the implementation, before the upgrading process from Xen 2 to Xen 3 is described in section 4.4.

4.1 Hardware Basis

At the beginning of the project, various tests [4, 5] have been performed in order to figure out which kind of hardware is necessary to virtualize the lab course shown in figure 4. The results have proven that no CPU bound bottleneck is suspected, but RAM seems crucial as 40 machines for the student work and some additional servers should be operated on one single host.

SuSE Linux filesystem images created by the YaST installer including tools for development, the graphical desktop environment KDE and some free disk space for the students' work are about 3 GB of size. Those plus additional disk space for backups have to be hosted on the server. Therefore, a SATA RAID using RAID level 1 to ensure the integrity of data is used.

The productive server is a Fujitsu-Siemens server with two AMD Opteron processors (model number 246 at 2.0 GHz), 4 GB of RAM (as the initial setup is using Xen 2 and thus only supporting 32 bit environments) and about 400 GB of Soft-RAID storage (RAID level 1).

4.2 Software Basis

To implement the virtual lab, Xen was selected among other virtualization tools. Its performance surpasses all other tools that can be used to provide virtual machines and network components when we started implementing the project in 2005. Ian Pratt demonstrates the performance

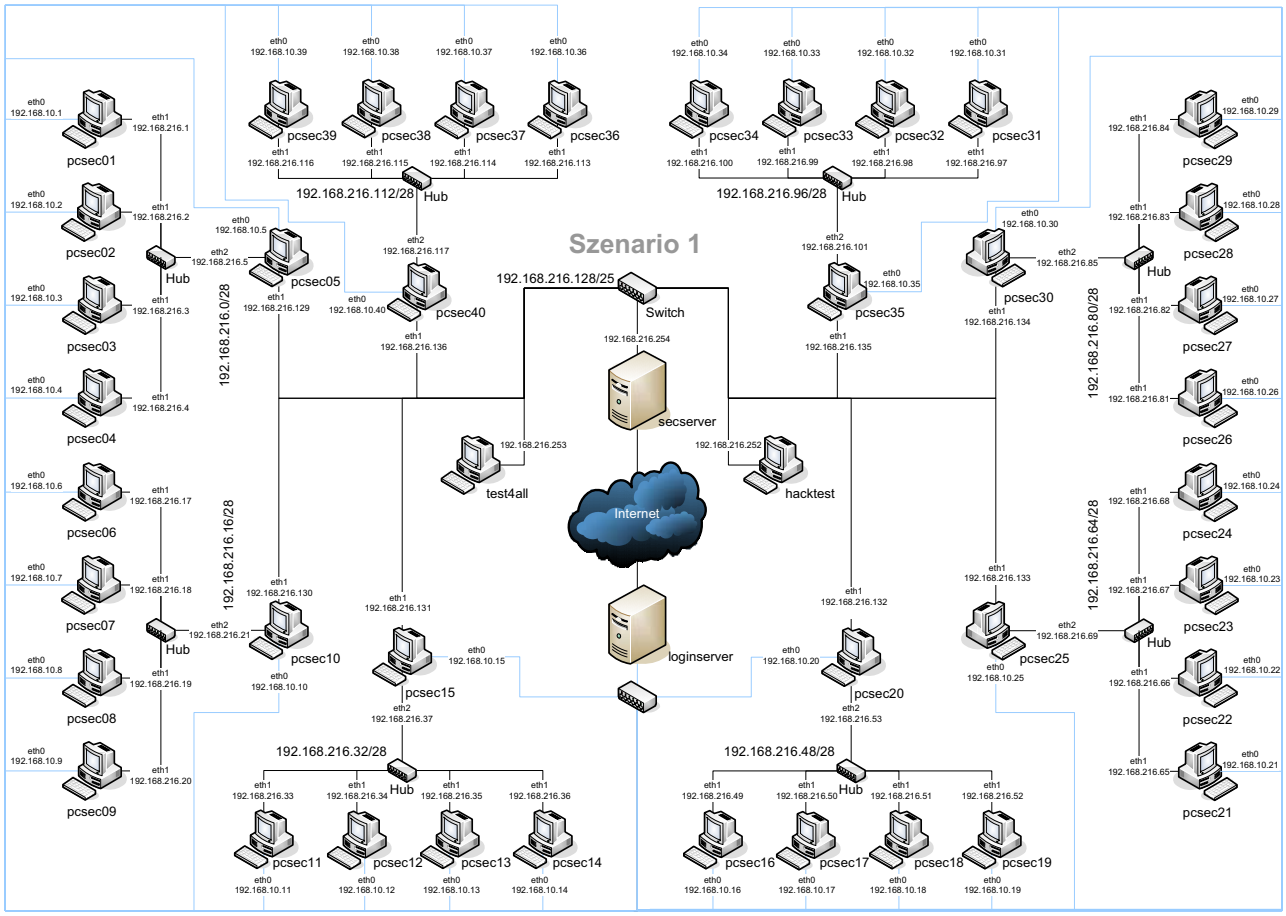


Figure 4. One of the virtual network environments

of Xen compared to VMware Workstation and User Mode Linux (UML) [7] in figure 1. The conclusion drawn in his work is that Xen performs best by far and in fact is close to a stand-alone Linux system.

Over and above this fact, Xen is very stable and implements a very powerful scheduling algorithm. As claimed by the Xen developers, it is possible to attack one virtual machine using DoS techniques, while other machines running on the same physical host are nearly not affected. As DoS attacks are executed during the course by students, this is an important fact which has been proven true.

4.3 Instantiation for the Lab Course

One of the scenarios used for the IT security lab course is shown in figure 4. The implementation of which is based on Xen version 2. This is due to the fact that Xen 3 did not perform well considering stability in our tests. It was still in beta stadium and thus the decision to deploy a version 2 system was made.

In this scenario, two switches, eight hubs, four servers, 40 student PCs and 94 network interface cards are needed. The darker marked interconnections between the servers and workstations depict the network topology used within the course (the "teaching network"), while the lighter connections represents the management network. The latter has to be deployed in order to grant access to the student machines as described in section 3.

To facilitate the use of graphical applications, X may be forwarded using SSH tunneling capabilities. This only proves suitable in case of the students working at machines with local area network connections to the system hosting the virtual lab. Besides, FreeNX [3], a remote desktop solution, is installed on every VM. FreeNX enables the export of the desktop environment in a very efficient manner. In our tests, even old-fashioned analog dial-up connections resulted in no overwhelming but acceptable performance. Both possibilities to use graphical interfaces can be used in combination with any of the two ways of connecting to the lab, either SSH tunneling or the usage of OpenVPN.

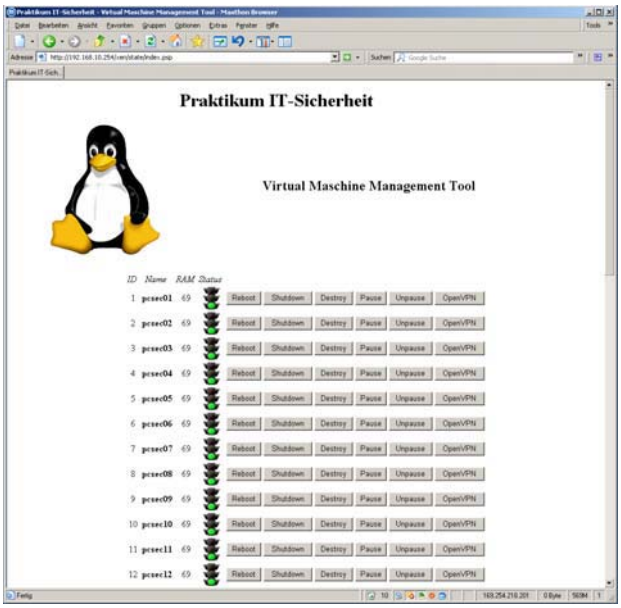


Figure 5. The management web interface

The possibility to access the interface of the Xen daemon *xend* using a web server via a python module enables the management of the virtual machines. Access control is realized using a custom-made authentication handler written in python. It compares any given password to the root password set in the virtual machine that shall be managed. Thus, access to both the virtual machine and the management interface is granted using just one single password. In this case, an apache web server in combination with the *auth* module is used and any traffic is encrypted using SSL. Figure 5 shows a screenshot of the home-grown simple management interface developed and used for the virtual lab course.

4.4 Upgrading from Xen 2 to Xen 3

New versions of SuSE Linux Enterprise Server do not support Xen 2 any longer and the demand to keep the software up to date is hard to satisfy. After operating the lab based on Xen 2 for two terms, a migration to Xen 3 was desired.

Some minor adoptions have to be made to the network configuration files and the configuration files used to create virtual machines. Both issues are more or less based on syntactical changes in Xen 3, resulting in minor problems.

In contrast, the port of the custom-made management tool to Xen 3 demands greater efforts because some interfaces of xend have changed. As a result, a part of the management tool has to be reimplemented using the new interface. Afterwards, the virtual infrastructure is working properly again.

Regarding the stability, both versions of Xen do not differ. Anyway, differences regarding the performance are obvious. While Xen 2 is a bit faster in general, Xen 3 is more powerful in accessing virtual disks using the new *xvd* (Xen virtual block device) driver. Both symptoms can be easily observed, e.g. by installing or booting new virtual machines.

One additional feature of Xen 3 is the possibility to virtualize Windows Workstation if suitable processors with the Intel-VT or AMD-V command sets are used. Our security course could thus be extended to Windows security issues as well. At the moment no suitable server hardware is available so that the course's focus lies on Linux. Anyway, the concept shown above still holds for Windows or mixed scenarios. Only some implementation issues would have to be adjusted as for example the current management platform just supports authentication and authorization methods for Linux VMs.

5 Experiences

Operating the virtual lab for four terms productively, no major problems occurred up to now. No problems related to stability are experienced, even critical actions like DoS attacks and malformed network packets sent during the lab course do not harm the infrastructure. Performance related problems are not noticeable, although 44 virtual machines are executed on one single physical host. Usually, load is distributed evenly over the week. Just in case of special events like tests or demonstrations that have to be passed as a milestone during the course, load is high, but the performance experienced by the end-user is still acceptable.

Compared to a native Linux machine a virtual Linux machine operated by Xen is insignificantly slower. Running more than one virtual machine at the same time is even more efficient. Due to intelligent scheduling algorithms, booting the virtual lab with all the virtual machines and services takes about 12 minutes. Hence, one single virtual machine needs about 16 seconds in average to start up into runlevel 5.

Network performance does not pose problems as well, even though many students are using graphical desktop sessions and all the traffic to the Internet is handled by only one physical network interface card. The network itself does not provide a bottleneck in the virtual lab. This also holds true for the virtualized network components interconnecting the machines among each other and providing the management network. Actually, the virtual network components in some cases show better performance than physical ones as they are simulated by kernel operations of the underlying host system.

The most important gain of the virtualization process is the reduction of administration costs to about a sixth part

compared to the initial course setup. Up to six advisors have been employed to manage and supervise the course and its attendees. Now, this task is accomplished by just one student advising the participants of the course regarding the content. As the hosting system now is a reliable server system, no disruptive incidents related to defects of hardware occurred yet. This was a frequent case before the virtualization of the lab and demanded a lot of in-time administrative work. Virtualizing the lab course, the hardware issues have been exchanged with the problem of managing VMs. Providing a management interface to students enabling reboots of hanging machines, etc. releases the teaching staff and shifts the efforts to the students while maintaining control. Additionally, access to the lab is possible from every computer connected to the Internet 24 hours a day. This leads to a maximum of flexibility in time and place for the students, especially as our lab course is offered at different universities.

6 Conclusion & Future Work

In this paper, a concept for a virtual IT infrastructure for higher education teaching is introduced. Security aspects, transparent usage of and convenient access to the infrastructure, as well as the comfortable management of the lab course are main requirements while designing the concept. The deployment as a proof of concept using Xen is a practical lab course dealing with IT security offered at the two Munich universities.

In everyday use, the experiences are predominantly good. The university's premises for the lab course could be released and valuable time of the teaching staff and administrators could be saved. This is mainly due to the fact that instead of managing several student PCs the management of one much more reliable server system has to be accomplished. The management of the virtual student PCs is performed by the students themselves, providing them with a web based management platform. In sum, this saved about two thirds of the costs to run the course.

In future work, the virtualization of other practical lab courses dealing with more technical content is considered. In this context, the question to which technical detail virtualization approaches seem applicable has to be answered. However, the concept presented in this paper has established a template for virtualizing other teaching environments.

Acknowledgment

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of this paper. The MNM Team founded by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, the University of Federal Armed Forces Munich and the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences. Its web server is located at `http://www.mnm-team.org`.

This paper was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[2] K. Begnum, K. Koymans, A. Lrap, and J. Sechrest. Using Virtual Machines in System Administration Education. In *Proceedings of 4th International System Administration and Network Engineering Conference*. System and Network Engineering, 2004.

[3] FreeNX Project. FreeNX. `http://freenx.berlios.de/`.

[4] T. Lindinger. Machbarkeitsanalyse zur Virtualisierung des IT–Sicherheit Praktikums. Technical report, Ludwig-Maximilians-University of Munich, Oct. 2005.

[5] T. Lindinger. Virtualisierung einer Praktikumsinfrastruktur zur Ausbildung im Bereich Sicherheit vernetzter Systeme. Master's thesis, Ludwig-Maximilians-University of Munich, May 2006.

[6] S. Liu, W. Marti, and W. Zhao. Virtual Networking Lab (VNL): its concepts and implementation. In *Proceedings of the 2001 American Society for Engineering Education Annual Connference and Exposition*, Texas, USA, 2001. American Society for Engineering Education.

[7] I. Pratt. Performance of xen compared to native linux, vmware and user mode linux. `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/performance.html`, Dec. 2004.

[8] I. Pratt. *Xen Status Report*. University of Cambridge, Dec. 2005.

[9] University of Cambridge. Computer Laboratory - Xen virtual machine monitor. `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`.

[10] University of Cambridge. XenoServers. `http://www.xenoservers.net/`.

[11] Universität Koblenz. Virtual Network User Mode Linux. `http://www.uni-koblenz.de/~vnuml`.

[12] XENSource. XenSource: Delivering the Power of Xen. `http://www.xensource.com/`.

Speichervirtualisierung

Storage Virtualisation, Basic Building Block for Dynamic IT

Simon Kastenmüller
Fujitsu Siemens Computers
Domagkstr. 28, 80807 München
Simon.Kastenmueller@fujitsu-siemens.com

Abstract

At a high level, virtualisation is an intermediary hardware or software layer that separates the logical view from the physical view to resources. Without virtualisation each server, operating system or application manages its dedicated physical resources. In a virtualised environment, the server, operating system or application do not need to know where the resource resides physically. Logical resources that are presented in a virtualisation layer replace server's, operating system's or application's physical resources. The virtualisation layer then dynamically maps these logical resources to the target physical resources, thereby increasing the flexibility and efficiency in resource assignment to a server, operating system or application. Virtualisation allows the IT department to separate decisions regarding server, operating systems and application from compute and storage devices and the associated management tasks.

Storage virtualisation can be realised in many layers of an IT infrastructure. Storage includes all layers beginning at storage arrays to storage network to storage-related layers in servers.

Storage virtualisation falls in the following main categories:

- *Storage array-based virtualisation (mainly block-oriented access)*
- *Virtualisation via NAS (mainly file-oriented access)*
- *Nearline virtualisation (also known as tape virtualisation)*
- *SAN fabric partitioning (virtual or logical SAN) and NAS partitioning*
- *LUN virtualisation via intelligent SAN switches*
- *virtualisation functionality bound to fibre channel switch ASICs*
- *In-band virtualisation appliances*
- *Server-based Volume Manager*

Emerging technologies

- *Virtualisation via grid-based file system is looked at as emerging technology*

Not every virtualisation layer has to be in place and not all combinations are possible due to restrictions in the relevant support matrices or due to other reasons. E.g. a connection needs not or in some cases even must not go from a standard LUN trough a VSAN trough ASIC-based virtualisation through a Volume Manager.

1. Storage Virtualisation in the IT Ecosystem

Storage Virtualisation has to be seen in a broader context within the data center infrastructure. The Dynamic Data Center (DDC) architecture from Fujitsu Siemens Computers describes this IT architecture that represents a breakthrough compared with existing paradigms. The focus of the DDC is on services that are provided to end users and the service level agreements (SLAs) that define the quality of provided services in terms of response times, availability, etc. The demanded quality should be offered at as low a price as possible (efficiency), and the IT in question should be able to adapt rapidly to changes in business processes (flexibility).

The DDC is therefore based on new hardware and software architectures that enable greater efficiency and flexibility, yet the very highest reliability. These architectures are complemented by concepts for integrating them in existing IT operations and running them efficiently.

Dynamic IT infrastructures must be developed on the basis of these new application architectures so that across-the-board successes can be achieved in IT efficiency and IT flexibility, yet stable operation is maintained. Ideally, every service can run on any system and be relocated between systems in a short time. The first step in this is to divorce the services

from dedicated hardware platforms via virtualisation. All hardware resources (servers and storage systems) are grouped in pools, from which they can be used flexibly as required. The traditional silo-like n-tier architecture is replaced gradually by a DDC architecture.

The operation of virtual and real resources is largely automated: "IT manages IT". If resources fail or are in short supply, corrective actions are initiated automatically by automation. Resources are automatically provided to services on the basis of predefined rules. These rules are based on the concrete business processes and resulting agreements with the users of the services and are specified in service level agreements (SLAs). Ideally, the services are charged according to degree of usage and required Quality of Services.

2. Driving forces for Storage Virtualisation

The overall goal of Storage Virtualisation is improving agility and Total Cost of Ownership (TCO). This can be achieved in different dimension whereas the different dimensions have interdependencies of some kind. The most important dimensions are:

- Better utilisation
- Manage heterogeneity
- New features on existing hardware

2.1. Better Utilisation

The first measure to improve utilisation is to provide resources via resource pools. It is not mandatory to work only with a single huge pool. In reality the resource pool will be divided into more than one pool according to the Service Level Requirements. In reality we won't see too many different resource pools. The number will start with two and level out at five as the differentiation between Service Level requirements will be then to small.

Based on the resource pools the virtualisation layer can then realise automatic load balancing. This will increase overall utilisation as the space limit for growth is defined once for the complete pool and not for each LUN in the pool.

Different pools with different Service Level agreements form the basis for autonomous transparent storage tiering. The rule engine will be provided by the integrated rule engine in an Information Lifecycle Management concept (ILM). The maturity of the integrations differs depending on the virtualisation layers.

2.2. Manage heterogeneity

As the virtualisation layers hides the physical attributes of the storage systems the storage systems behind the virtualisation layer can be from different suppliers. Most of the management will be done in the virtualisation layer so that one management tool can manage heterogeneous storage.

This applies also for technology refresh. In these cases it is necessary to migrate data from old storage systems (probably with a different architecture) to newer storage systems. The virtualisation layer can hide these differences and can transparently migrate data.

2.3. New features on existing hardware

Each storage system has some limitation e.g. maximal number of LUNs, maximal numbers of SnapShots etc. Modern storage virtualisation solutions not only virtualise the LUN presentation but they offer all the extended storage functionality such as SnapShots, replication etc. By moving these functions from the storage arrays into the storage virtualisation layer it gets independent from the limitations of the storage systems in the back-end. As a by-product these functions will then work across heterogeneous storage systems. On the other hand it does not make sense using the extended storage functionality in the virtualisation layer and in the back-end in parallel. Therefore expensive functionality on existing storage in the back-end may lose its value when moving this functionality into the virtualisation layer.

3. Virtualisation layers

In the storage area we can identify many points where virtualisation can take place. At SNIA we can find a systematic view on these locations. SNIA differentiated the following areas.

- What is created
- Where is it done
- How is it implemented

3.1. What is created

This area falls into the following sub-blocks: "Block virtualisation", "Disk virtualisation", "Tape, Tape Drive, Tape Library Virtualisation", "Filesystem / File / Record Virtualisation" and "Other Device Virtualisation". Most of these points are self explanatory. Some parts such as "Object Storage Devices" have to be linked to a sub-block. This could be either "Disk Virtualisation" or "Other Device Virtualisation".

3.2. Where is it done

At a high level view SNIA differentiate between "Host-based / Server-based Virtualisation", "Network-based Virtualisation" and "Storage Device / Storage Subsystem Virtualisation".

3.3. How is it implemented

Here we have only two blocks: "In-band Virtualisation" and "Out-of-band Virtualisation". With the latest development we should add "Split Path Virtualisation".

In-band Virtualisation means that the Virtualisation layer sits in the data path. All I/O have to travel through this layer. The main advantage is the single point of virtualisation. On the other hand this can lead to scalability issues. In the out-of band model the virtualisation layer is outside the data path. The I/O path stays as it was before. This model provides the best performance but normally requires special drivers in the host- / server systems. Split Path Virtualisation makes use of intelligent switches. Redirection etc. is done in the switches. The virtualisation management is outside the switches. There is only a small additional latency due to the redirection etc. in the switches. In general this solutions scales as a normal switch architecture would scale.

4. Storage virtualisation stack

This above mentioned SNIA Model gives an good overlook but when we have a closer look we can identify more virtualisation points.

Partitioning of hardware or software resources is also a kind of virtualisation. Partitioning can be realised in the storage arrays, in the tape libraries or on the switches. Partitioning generates two or more separated views which have separated resources on the same basic hardware.

ILM in the box is another example of virtualisation. The clients will see only one interface. However the system will internally deploy different tiers for data storage. The ILM level can be realised on block levels. For instance the system will then move e.g. rarely accessed blocks from one tier to another tier in the same system. More common is the ILM in box for arching systems. Complete files would then be migrated from disk storage to tape storage in a transparent way.

A next level is the so called spindle virtualisation. Presenting LUNs (Logical Units) has been an established technology for years. On top of the LUNs we have the different RAID-levels. In this area we can also position thin provisioning. This means that the client sees more space than is actually physically allocated. The physical allocation takes places when the customer writes the first time into an area. The advantage is that one can logically allocate the maximum e.g. to a file system and there is no need to expand the file system. On the other hand this needs careful planning as over-allocation can happen.

In the next block towards the server network-based virtualisation comes into play. This functionality is already defined in the SNIA model.

A relatively new virtualisation layer is the host channel adapter virtualisation. N-port ID virtualisation realises many logical I/O channels between the switches and the host / server systems. This can reduce especially the number of up-links into the network as more than one connection can be established on the up-links.

Going up in the storage virtualisation stack we have the virtual volume manger. In principle it is the same idea as with spindle virtualisation but on a higher level. LUNs can be sliced or concatenated and volume manager normally offer different RAID levels. Volume manager can work on heterogeneous storage. Volume manager normally have only the view of one host / server system but this will change in future. Volume manager implementations on different operating systems differ substantially.

Grid-based file systems are on top of the virtualisation layer. At the level of Grid-based file systems one can differentiate between clustered file systems and parallel file systems. Cluster file systems store the metadata distributed on back-end storage whereas on parallel file systems normally the metadata are managed by special nodes (metadata server).

Partitioning, ILM in the box and spindle virtualisation is realised in the storage systems itself. Storage network-based virtualisation is either realised via special appliances or it makes use of intelligent fibre channel switches. Host channel adapter virtualisation, volume managers are normally deployed on the host systems. Grid based file systems consist of more than one storage node. The physical storage is either direct attached storage in the nodes or external storage which attached via fibre channel or iSCSI.

5. Virtualisation pro and cons

5.1 Pros

In virtualised environments one can mix and match different storage arrays. They can be from different vendors. Virtualisation breaks the vendor lock-in in this area. The storage can also incorporate storage with different quality of services. The storage virtualisation layer will then pool the different quality of service area and provide this storage via different pools to the users. If the higher functionality is moved into the virtualisation layer then cheaper storage can be deployed at the back-end. The virtualisation layer provides a consistent feature set in heterogeneous environments and a single point of administration to a certain extent. In total this improves agility and TCO.

5.2. Cons

In mostly all implementations the virtualisation layer introduces an additional management and monitoring layer as the management of the storage can only moved to the virtualisation layer to a certain extent. Error diagnostics or physical extensions normally have to be carried out in the storage systems itself. With in-band implementations the scalability issue has to be

looked carefully. If out-of band virtualisations are based on host-based agent then software dependencies have to be considered which is harder to manage. Split-path and in-band implementations have to be considered as new storage systems. This requires certification and qualification.

6. Block-level Virtualisation

At the block-level virtualisation one has to differentiate between in-band, out-of-band and split-path implementations. In-band virtualisation implementations are normally based on appliances which can even consist of standard server hardware. As each and any I/O has to travel through this appliance scalability is an issue. On standard server hardware the bus systems will be most likely the limiting factor. The out-of-band implementation does not introduce new hardware into the data path. Scalability should be the same as if there were no virtualisation layer. The management of the virtualisation layer will then be realised in a separated instance outside the I/O-path. Array-based virtualisation implementations are hard to classify into these categories. At array-based implementation heterogeneous storage or storage arrays are attached to the array with the virtualisation layer.

Block-level virtualisation is not always completely transparent. This should be discussed on the example of thin provisioning. If the storage resource management is only deployed on the host systems it will see the total storage as available even if it is only logically attached. Real benefits of thin provisioning are only given if the application stack can work with this feature (e.g. Autoextend with Oracle).

Block-level virtualisation differs substantially in the enterprise and entry market. In the enterprise market functionality and high availability are the dominating factors. In the entry market many projects are faced with integration issues. Customers seek for new base functionality for existing old storage at low price points.

7. File-level virtualisation

File-level virtualisation is mostly based on the NFS or CIFS interface. Beneath the virtualisation layer file servers or NAS systems form the basis. This virtualisation is most successful in the high-end segment. Transparent migration of files is the most cited value proposition. First placement and load balancing come next. Managing a Global Name Space is a mandatory functionality for transparent migration, load balancing, first placement etc. File-level virtualisation supports consolidation, tiered storage and provides a single point of management for heterogeneous systems. In principle there exist two different implementations. The file-level virtualisation layer is either implemented as an appliance (often on adjusted network switch hardware) or is a pure software solution which manages an external Global Name Space.

8. Nearline Virtualisation

The value proposition of nearline virtualisation is clear and easy to understand not only in the mainframe area but also for open systems. Nearline virtualisation is not a nascent market. Nearline virtualisation is based on proven technology. Fujitsu Siemens Computers' CentricStor is a de-facto standard.

CentricStor is a powerful virtual magnetic tape solution which enables fully integrated 'disk-to-disk-to-tape' data backup for all corporate data.

CentricStor is implemented as a transparent virtualization layer between mainframes, servers and magnetic tape systems. CentricStor has a virtual interface to the servers and so supports many servers, operating systems and software programs for data backup like no other solution.

With CentricStor, you can consolidate many magnetic tape archives simultaneously and thereby reduce overall running costs (TCO) because you need fewer tapes, drives and magnetic tape systems. CentricStor also dramatically shrinks the time windows required for data backup and at the same time speeds the recovery of data from business-critical applications.

Unlike conventional monolithic 'virtual tape libraries', CentricStor is what is termed a 'virtual tape appliance' (VTA) which uses a flexible, modular CentricStor grid architecture (CGA) and true magnetic tape virtualization, also called 'true tape virtualization' (TTV).

CentricStor offers unprecedented scalability, connectivity and data recovery following a system crash, as well as a substantially reduced total cost of ownership (TCO) in terms of data backup and restore, business continuity, HSM and batch processes. CentricStor also supports a wide range of service level agreements (SLAs) which are necessary when an organization wants or is required to align its data storage with an information lifecycle management (ILM) strategy.

All this makes CentricStor the only virtual magnetic tape solution with which magnetic tape technology is made suitable for ILM concepts.

9. Virtualisation on Grid-based file systems

Traditional storage areas consist of limited storage processors. In most cases the array can be equipped with two storage processors which form a high availability cluster. This concept limits the throughput as each I/O needs some resources on the storage

processors. Grid-based file systems however scale not only in direction of capacity but also in the direction of throughput. In theory Grid-based file systems don't have limitations in the number of nodes. In reality there are of course limitations. The reasons are test time, test resources, type of implementation as clustered file system or parallel file system etc. The communality between the implementations is that most of them can run on standard server hardware. There are many implementations on the market (in the range of hundreds). Comparing the different implementations is difficult as there are many requirement dimensions. Following are some examples:

- Scalability (# of nodes, single file system or maximum number of files systems)
- Virtual filers which can be easily moved between nodes
- Based on industry standard servers or on special hardware
- Type of interconnect between the nodes
- Cluster RAID (wide stripping), RAID-level, rebuild architecture in the case that one or more disks fail
- Software availability (snapshots, thin provisioning, local / remote replication)
- Management concepts / management interfaces, dynamic expansion / reduction of nodes
- Front-end: Interfaces (NAS, iSCSI, FC)
- Back-end: DAS or external storage.

10. Server & storage virtualisation working together

The above discussed virtualisation stack shows the most important virtualisation layers. In reality one will make use of only a few selected virtualisation layers. On the other hand we have the server virtualisation which needs storage resources. Server and storage virtualisation have to work together. Server and storage virtualisation has also to work together with network virtualisation but network virtualisation is not covered in this paper. Following we explain some examples where server and storage virtualisation work together.

10.1. Blade servers

Normally each blade server needs storage (and network) connection to the infrastructure. In a rack with many hundreds of blade servers this lead to high number of necessary ports on the edge switches. With blade switches supporting the N-Port IP virtualisation the necessary up-links to the edge switch can be reduced as more than one logical connection can be transported over a physical up-link.

10.2. Server virtualisation under VMware

The next point for improvement can be seen in the storage addressing scheme in fibre channel and VMware environments. Normally the ESX system holds the World Wide Name (WWN). In the next iteration of VMware together with N-Port ID Virtualisation it will be possible that each guest system will see its own WWN. This improves security so that even more applications can be moved into virtualised server and storage environments.

Working with VMware often means that the storage of the guest systems is virtualised in the VMware File System.

The first option for doing backup is running the backup jobs in each guest system. The advantage is that all the existing features can be used which can put the applications into consistency states (transaction consistency). On the other hand backups consume a lot of resources and can only use such backup devices which are supported by the ESX server.

Another option would be running the backup on the ESX server. This option is very limited as the backup software needs qualification by VMware.

To improve this situation VMware introduce the Consolidated Backup (VCB) which can be carried out from a separated backup server. The advantage is that range of supported devices is much broader as on the ESX server. Additional the method opens easy file by file recovery especially in heterogeneous Windows environments.

10.3. Provisioning

Dynamic Data Center solutions work with autonomic cycles. Based on a storage pool the autonomic cycle monitors the configuration, analyses status, adapts the configuration based on rules and the information which were gathered in the previous steps and executes the necessary actions. Dynamics means that storage space can be provisioned dynamically. For example if in the autonomic deploys new / additional servers the new servers needs storage accordingly. This process is managed via provisioning. Normally complete LUNs are provisioned by the autonomic cycle but this is not limited to this. Another example would be CIFS or NFS shares. Storage Virtualisation will help dramatically to improve the flexibility as the entities can be taken out of a storage pool even in heterogeneous environments.

10.4. Transparent Migration

With the technology available today each configuration has to face the problem of technology refreshes. In traditional configurations this was managed by unloading data from the old storage system to backup media and afterwards loading the data from the backup media to the new storage system. The main disadvantage was that this needs exclusive

time windows during this time the application could not work in update mode. With the increased requirements these windows are no longer available. Thanks to the virtualisation layers available today these migration process can be done without application interruption especially when the data already have been put under the management of the virtualisation layer. All further migration processes will then run with no application interruption. One example for this technology is CentricStor from Fujitsu Siemens Computers. The "True Tape Virtualisation" will shield the application layer completely from the physical layer. Technology refreshes or in general term "Migration" is no longer a SLA (Service Level Agreement) issue.

11. Summary

Virtualisation via partitioning is the most transparent approach and it is widely accepted

ILM in the box is an interesting approach but the solutions on the market address special market segments such as archiving. Maybe we will see more solutions in future.

Presenting disks as LUNs has been an established technology since years. The same is true for NAS systems. In the area of LUN virtualisation there are some extended features such as thin provisioning.

For block-level virtualisation a new battle will start for split-path, in-band and array-based virtualisation. Special segments are addressed with host-based virtualisation especially in the high-end Unix-market. Many projects in the entry segment for block-level virtualisation are faced with integration of existing storage (e.g. new live / features into old storage.

File-level virtualisation delivers interesting features for transparent migration, load balancing or first placement if it is implemented as in-band functionality.

Nearline virtualisation is an established market with well understood ROI. Fujitsu Siemens Computers' CentricStor is the market leading solution.

Using commodity hardware components in Grid-based file system solutions is getting more and more attractive. This emerging market will start in special segments and enlarge into enterprise market.

OSD (Object Storage Devices) is a future market. In principle controller hardware will provide enough resources to realise OSD but this has to be integrated / adjusted with I/O stack above which needs a lot of work to be done.

All these virtualisation layers support the Dynamic Data Center concept with its autonomic cycle. Combining Server Virtualisation, Storage Virtualisation and Dynamic Data Center concepts helps customers to makes huge progress in the direction of Service Oriented IT infrastructures.

Storage Cluster Architectures *

André Brinkmann
University of Paderborn
brinkman@hni.upb.de

Sascha Effert
University of Paderborn
fermat@upb.de

Abstract

Storage clusters try to transfer the idea of cluster computing into the storage domain and to scale capacity and performance by simply adding new cluster components. This paper presents the architecture of storage clusters and presents analytical considerations on the scalability of storage clusters and presents a storage cluster architecture based on peer-to-peer computing. It is shown that storage clusters are able to scale up to hundreds of servers and clients. The storage cluster environment has been successfully implemented and tested on a Linux based HPC-cluster. The measurement results presented in this paper demonstrate the feasibility and scalability of this architecture.

1 Introduction

Cluster-based storage tries to transfer the idea of cluster computing into the storage domain. A storage cluster is based on a set of storage appliances, called storage bricks, which work together closely and can be seen, from the outside, as a single, huge and fast storage system. The storage bricks are managed by a storage cluster middleware that is implemented as a software system managing the distributed state information about the storage cluster [17].

A major distinction between storage clusters and conventional storage architectures is that storage bricks are assembled based on commodity server architectures, enabling cost-savings compared to dedicated architectures [3]. Therefore each storage brick does not only provide storage capacity, but also computing and communication power. The computing capabilities enable a storage brick to contain a software management stack and to act as a storage appliance. The software stack inside a storage brick is responsible for a seamless integration of the brick into the cluster environment. Integrating a new storage brick therefore only involves the assignment to a storage resource pool, all other administration tasks, like authentication or rights management, are handled by the middleware. A characteristic element of storage clusters is that adding new storage bricks does not only increase storage capacity, but also the performance of the entire cluster.

Storage bricks can either use directly attached storage or networked storage as persistent storage. In the first case, only an interconnection infrastructure between the nodes of the storage cluster and to the client systems is necessary to provide scalable storage. In the second case, the bricks have to be connected to the networked storage systems over a storage area network, inducing additional costs and complexity, but also enabling the bricks to share storage devices without communication between the bricks.

The idea of a storage cluster as a collection of smaller components is closely related to storage virtualization and has been implemented first in the Petal prototype [13]. The main task of a storage cluster is to hide the complexity of the underlying storage systems by using a block-based storage virtualization environment or a distributed file system [20] [16] [9]. An example for an academic storage cluster is Ursa Minor, which provides access to objects instead of files or blocks and which is able to change data encoding and therefore performance and reliability of data objects based on attributes and access patterns [1]. The aim of the *Federated Array of Bricks (FAB)* is to deliver enterprise properties from a set of storage bricks at a fraction of the costs of an enterprise storage array [19]. The *V:Drive* project is based on randomized data distribution schemes, which are able to evenly spread data and accesses among all participating bricks and offer fast reorganization in case of failures or the integration of new bricks [6].

Storage clusters often use Ethernet as interconnection technology to the clients and between the storage bricks. In this paper we will focus on *Internet SCSI (iSCSI)* as interconnect protocol, which has been developed as an extension of the SCSI protocol environment for TCP/IP based networks [22]. Additional block level storage protocols over Ethernet are *HyperSCSI*, *NBD*, and *ENBD* [23][2]. It is of course also possible to use high speed networks like Infiniband or Myrinet as interconnect between the bricks or to the

*This work has been partially supported by the EU within the 6th Framework Programme under contract IST-511531 *Highly Predictable Cluster for Internet Grids (HPC4U).*

clients.

Inside this paper we investigate the sub-class of storage clusters where storage bricks use directly attached storage devices as persistent storage with a block-level interface. Furthermore, we assume that clients are not allowed to load proprietary drivers to support access to these devices. This is a key requirement for building open systems RAID-System, which are offered by vendors like Equallogic or LeftHand Networks [8].

The requirements concerning this storage cluster architecture differ significantly from approaches like Petal, where clients load an additional module that gives hints about the data location, and it differs from Ursa Minor that is based on the concept of object storage devices, where accessing clients also know where to access data. The architecture of the FAB-project is closely related to the architecture used inside this paper, but the publications do not consider the influence of the interconnection network between the peers on scalability.

The performance of this sub-class of storage clusters mainly depends on two different aspects: The ability to evenly spread data blocks and requests to the data among the storage bricks and the communication overhead between the peers. The communication between the peers is especially important, if the hard disks are as fast as or even faster then the communication links. This can occur if a set of disks inside each storage brick is used as internal RAID environment and the access pattern is sequential or if solid-state disks are used as persistent storage. Communication between peers is always necessary, if a peer needs to access data that is stored on another peer.

After giving a introduction into the system architecture and cost aspects in section 2, we analyze the influence of inter-node communication on the scalability of the network in section 3. The calculations are based on the assumption that the interconnect is the bottleneck of the network and we show that the internode-communication has got a significant influence on the performance of a storage cluster.

The analytical results of this paper are complemented by measurement results for scalable storage clusters in section 4. The measurements have been performed on a high performance computing (HPC) cluster environment under Linux. Based on a storage cluster architecture that has been composed from publicly available components and the cluster volume manager V:Drive we show that the analytical results fit very well with reality. We will present the measurement results for up to 24 cluster nodes and 24 client nodes including data replication schemes. Part of this work has been previously published in [3] and [4].

2 Technology

Standard server and interconnection architectures have become powerful enough to compete with dedicated storage architectures. Two important aspects of storage cluster hardware are the interconnection network between the nodes of the storage cluster and the connection between the nodes and the persistent storage. From the software perspective, the most important performance challenge is to ensure an even balancing of the data request among the nodes. Otherwise, peak demands can (and will) endanger system scalability.

In the following, we will discuss the basic hardware and software components of a storage cluster and their influence on costs and performance. After presenting the overall system architecture, we focus on technologies for building storage clusters based on direct attached storage devices (DAS). The scope of the section Cluster Interconnects is on interconnection technologies between the cluster nodes.

2.1 System Architecture

Storage clusters are built as peer-to-peer solutions, where each peer / brick is a standard server system. Each client of the storage cluster can be connected with an arbitrary storage brick and each request can be served by every brick (see Fig. 1). The connection between clients and storage cluster is normally based on Gigabit Ethernet; interfaces can either be a block level interface, like iSCSI, or a file interface like NFS or CIFS. If the bricks are connected to the disks via a storage area network (SAN) then the only communication between the peers involves the exchange of meta information. In our case, the bricks utilize their internal storage. To ensure that each brick can access every data block, also bulk data has to be exchanged between the peers. This data exchange can be done via a dedicated high speed network, like Infiniband or Myrinet, or in our case via less expensive standard Ethernet connections.

As major software component, a virtualization layer has to ensure that all bricks have got the same, consistent view on the data. This virtualization can either be performed by a block-level storage virtualization or by a distributed file system. Standard server architectures are less reliable than dedicated storage systems and the virtualization layer therefore has also to ensure that data is protected against the failure of one or more storage bricks.

2.2 Storage Interconnects

The different technologies for the interconnection of server nodes and persistent storage can be distinguished in two categories. Direct Attached Storage (DAS) describes

storage systems, which are directly connected with a single server node. Opposed to DAS environments, the storage systems inside a Storage Area Network (SAN) are connected with the servers via a dedicated physical network. In a SAN, a large number of servers can be connected with the same physical storage systems. As we will see in subsection SAS and SAS-expander, the new serial SCSI technology is building a bridge between the concept of direct attached storage and storage networks and tries to mix the best properties of both approaches. In this paper we will concentrate on ATA, SATA, SCSI, and SAS, which are used as direct attached storage systems inside storage bricks. For a description of SAN-storage systems, see e.g. [7].

ATA and SATA: The Advanced Technology Attachment (ATA) standard defines a set of interfaces for the direct interconnect between storage systems, like hard disks or CD-ROMs, and computer systems. Besides the standard definition as ATA, the technology has also been named as Integrated Drive Electronics (IDE) or Enhanced IDE (EIDE). The name IDE and EIDE is pointing to the controllers, which have been integrated into the storage systems.

To overcome the drawbacks of parallel ATA (PATA) at higher frequencies and cable lengths and to enable higher data throughput, Serial ATA (SATA) has been introduced in 2003. The first generation of SATA interfaces supports frequencies up to 1.5 GHz and uses a Low Voltage Digital Signalling (LVDS) scheme. Using differential signal transmission, a signal change triggers both used wires to change their voltage level at the same time, but with different phases. LVDS is able to substantially reduce interference liability and enables the increase of wire length to 1 m while simultaneously boosting the data transfer rate. Besides the 1.5 GBits/s standard, the 3 GBits/s standard and corresponding storage devices are already available.

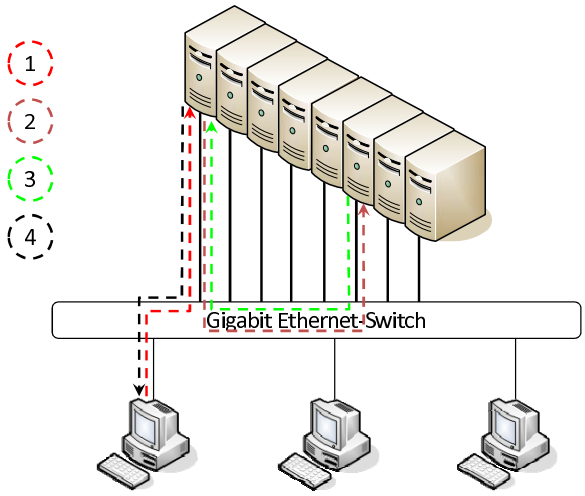


Figure 1. Storage cluster architecture.

Important technical innovations of the standard are hot swapping of storage devices and the introduction of Native Command Queuing (NCQ). NCQ enables reordering of requests inside the storage systems to increase data throughput [11]. Furthermore, current SATA hard disks posses better reliability and higher meantime between failures. The integration of advanced error detection schemes, queuing mechanisms and the elimination of jumpers enable the usage of SATA hard disks in web servers or in RAID environments [14].

SCSI and SAS: The Small Computer System Interface (SCSI) is the eldest of the addressed storage interconnects. Classical SCSI hard disks use a parallel interconnect for the data transfer and distinguish between dedicated data and control wires. The SCSI standard is able to address up to 4 (later 16) devices, which are connected in a daisy chain.

SCSI has become much more than a standard for host bus adapters that can interconnect hard disks and computer systems. The upper level defines a set of commands, while lower levels define interconnect technologies. The interconnects range from classical, parallel SCSI to serial data transfer technologies, like Fibre Channel, FireWire, Serial Attached SCSI, and even to the iSCSI protocol, which enables the transport of block oriented SCSI commands over the Internet.

Serial Attached SCSI (SAS) starts to replace present parallel SCSI interfaces on the physical transmission layer. Parallel SCSI reached its physical limits with the current Ultra-320 SCSI standard and has to cope with similar problems such as parallel ATA interfaces. The changeover to serial transmission technologies should overcome the problems of different signal transmission times on long wires and of crosstalk at high transmission frequencies [21].

In contrast to parallel SCSI interfaces, SAS defines point-to-point connections between devices with a connection speed of up to 3 GBit/s (net performance 300 MByte/s) in each direction and for each physical interface. To enable redundant architectures, SAS envisions dual porting in SAS hard disks, where each port has got a different address. This means that each SAS hard disk has got two ports and each port can be assigned to a different host bus adapter. To overcome the limitations of a single host bus adapter, SAS introduces the concept of fanout and edge expanders, which are the foundation of SAS networks with up to 16256 SAS devices in a single environment. At most one fanout expander is allowed in an environment [18].

Based on their simple interconnection topology and the fact that each SAS port is only allowed to be interconnected with a single SAS address at each point in time, SAS is not classified as a storage area network. An interesting feature

of SAS controllers is that they support direct attachment of native SAS hard disk as well as the integration of SATA disks. The attachment of SAS hard disks to a SATA controller is not possible.

Comparison of disk technologies: SATA and SAS devices have inherited many properties of their ancestors ATA and SCSI. E.g., many SATA devices are just IDE disks with a new interconnection interface. The pricing of SATA devices is therefore similar to the pricing of IDE disks (see Fig. 2(a)). Requiring only a new interface chip, the costs per GByte of a SATA disk differ only marginally from an IDE disk of the same size. There are only very few deviations from this behavior, where some SATA disk are priced in a range between 2 Euro/GByte and 4 Euro/GByte[1].

Examining Fig. 2(b) and Fig. 2(c) it becomes obvious that these disks differ both in speed and reliability from standard ATA devices. In both cases, former enterprise-class SCSI hard disks have been redesigned with a new SATA interface. Afterwards, these disks have been positioned between SATA and SCSI hard disks and are an interesting alternative for building reliable disk environments. Comparing SAS devices with SCSI and FC devices, it can be observed that there is a significant price gap. SAS technology is still that new that SAS drives are getting a price offset compared to standard enterprise class devices. It can be foreseen that SAS devices will become comparable in pricing with SCSI and FC devices over the next few years.

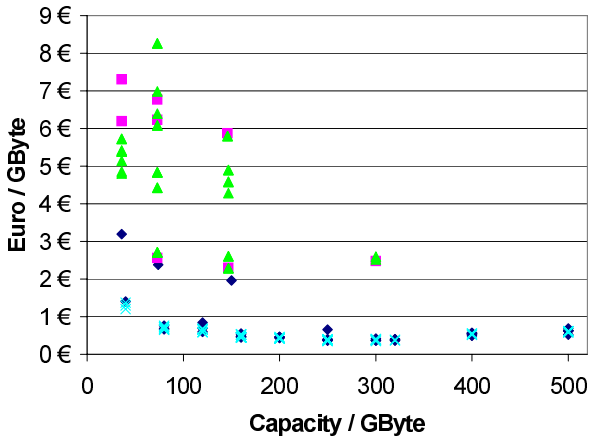
It is important to notice that the meantime between failure (MTBF) of ATA/SATA and SCSI/FC/SAS devices can not be directly compared. Enterprise class disks are tested under the assumption that they are accessed 24 hours a day, ATA/SATA devices are assumed to be accessed only 25% of the time. Increasing this on-time can lead to a significant decrease of the MTBF, leading to less reliable environments.

2.3 Cluster Interconnects

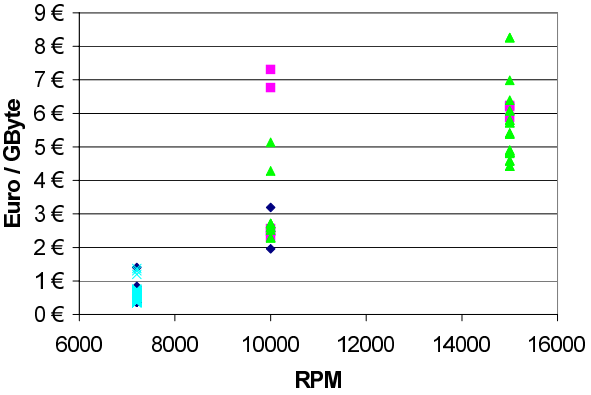
The nodes of a storage cluster, the storage bricks, can be connected with each other via different network technologies. These interconnects between the nodes differ in terms of bandwidth, latency, and cost. Nevertheless, the scalability properties of a storage cluster are significantly influenced by the physical properties of the interconnection technology between the nodes.

From the perspective of the physical implementation of the interconnection technologies, the different approaches for cluster environments become more and more similar. All technologies offer or will offer in the near future a bandwidth of 10+ GBit/s. Differences can be seen on the protocol layer. Ethernet is closely related to the TCP/IP protocol

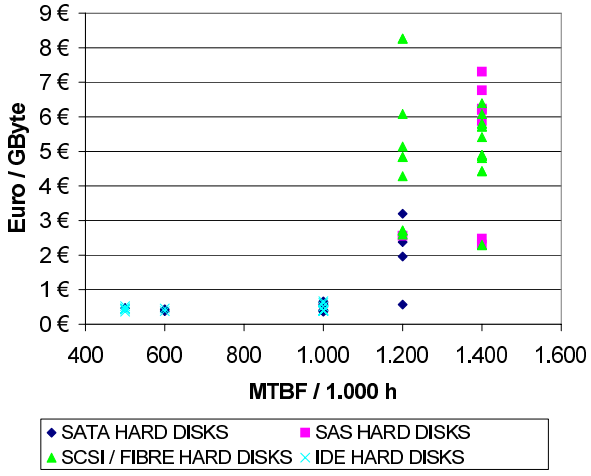
[1] All cost estimates inside this paper are from February 2007.



(a) *Costs per GByte*



(b) *Costs vs. RPM*



(c) *Costs vs. MTBF*

Figure 2. Cost comparison for different disk technologies

1U Server				
Modell	Setting	Hard Disks	Price	€ / Gbyte
Vendor A 1U Server	1 x Dual Core Intel Xeon 1,6 GHz 3ware 9550SX-4LP 4 channel RAID controller 4 x Gigabit Ethernet, 8 GB RAM 5 years warranty	4 x 250 Gbyte SATA	4.796,00 €	4,80 €
		incl. mgmt. software	5.000,00 €	9,80 €
		4 x 500 Gbyte SATA	5.316,00 €	2,66 €
		incl. mgmt. software	5.000,00 €	5,16 €
		4 x 750 Gbyte SATA	5.956,00 €	1,99 €
		incl. mgmt. software	5.000,00 €	3,65 €
2U Server				
Modell	Setting	Hard Disks	Price	€ / Gbyte
Vendor A 2U Server	1 x Dual Core Intel Xeon 1,6 GHz 3ware 9550SX-8LP 8 channel RAID controller 4 x Gigabit Ethernet, 8 GB RAM 5 years warranty	8 x 250 Gbyte SATA	5.531,00 €	2,77 €
		incl. mgmt. software	5.000,00 €	5,27 €
		8 x 500 Gbyte SATA	6.571,00 €	1,64 €
		incl. mgmt. software	5.000,00 €	2,89 €
		8 x 750 Gbyte SATA	7.851,00 €	1,31 €
		incl. mgmt. software	5.000,00 €	2,14 €
Dedicated iSCSI Server				
Modell	Setting	Hard Disks	Price	€ / Gbyte
Vendor A iSCSI RAID	Dual Controller, 4x 1 Gigabit Ethernet 2 Gbyte Cache	15 x 500 Gbyte SATA	11.915,00 €	1,59 €

Figure 3. Cost comparison of different server configurations.

stack that is used in local area networks and in wide area networks. Based on the protocol and software overhead of the TCP/IP protocol, latencies of Ethernet interconnects are still a magnitude higher than latencies of dedicated cluster interconnects, like Myrinet or Infiniband.

2.4 Cost Scaling of Storage Clusters

Storage costs are composed from a variety of different parts. Besides the investment into storage hardware, they also include software, integration, management, and additional costs and can be distinguished into one-time costs and recurrent costs . Besides the initial hardware costs, only the initial software costs can be easily identified. Most out-of-the-box environments, including pre-integrated storage clusters, can be installed within a few hours from most administrators. On the other side, building a storage cluster based on open or closed source software for the first time from scratch can take many weeks or even months before the environment is ready to be used in a productive environment, leading to storage costs much higher than the initial hardware costs. This would contradict one of the major advantages of storage cluster architectures, their simplicity in design and scalability.

The costs of a storage cluster can be influenced by the overall capacity, speed, and reliability of the environment. Inside this paper, we only analyze Gigabit Ethernet based server architectures from a single vendor, which are either equipped with SAS or SATA devices.

Fig. 3 compares two different servers with a dedicated iSCSI storage server. The price comparison shows the influence of the server price on the price per GByte of a storage brick. Being equipped with complete server hardware, the offset of the base system is much bigger than the price of the hard disks themselves, forcing a storage brick to integrate as much hard drives as possible. The 1U server is *only* able to integrate 4 hard drives and therefore has to cope with high costs for the processor, RAID controller, and cache RAM. The influence of this offset becomes even higher if the internal HW RAID-controller uses one of the disks as parity disk inside a local RAID group and therefore the usable capacity decreases by 25%. It is interesting to observe that the dedicated iSCSI server has nearly the same costs per GByte as a 2U server with 8 750 GByte disks, especially as the dedicated iSCSI server, which can not be scaled up to form to a storage cluster, is build up from standard server hardware and the disk density has been increased to 15 hard drives. This gives an insight into the cost calculation for the storage software, which includes in this case snapshot functionality and the iSCSI interface. More detailed examinations of cost factors in storage cluster architectures can be found in [3].

3 Communication Overhead between Peers

Scalability inside a storage cluster is bounded by a number of factors. Important aspects concerning the scalability are the ability of the data distribution to balance data and requests among the peers and the communication overhead between the peers that is induced by the exchange of data and information between the peers and the underlying network technology. In this section we will focus on communication between peers and we will assume that this kind of communication is only necessary to exchange data blocks and that only small amounts of metadata have to be exchanged between peers. Furthermore we will assume in a first step that the underlying data distribution scheme is able to evenly distribute data among the peers, so that all peers are able to participate according to their storage capacity and performance. An even distribution of accesses can either be achieved by striping the data over the peers or by using a distributed hash function that randomly distributes data over the peers [12] [6] [10].

Communication between two peers is necessary if a peer needs to read or write data stored on another peer. Figure 1 depicts a typical read inside a storage cluster. A client is connected to one storage brick inside the cluster that acts as iSCSI target for this client. This storage brick will be called master peer for this client in the following. In a first step, the client sends a read request to its master peer. If the master peer does not store the corresponding data block, it has to forward the request to a peer in step 2. The peer reads the data block from its cache or disk subsystem and returns the data block in step 3 to the master peer. In a last step, the master peer sends the resulting data block to the client.

This process does not only involve the forwarding of

control messages between the peers, but also the movement of bulk data from the peer containing the data to the master peer. This movement seems to be unnecessary, because this data block could be (theoretically) sent directly from the data source to the client. Unfortunately, this is not possible for iSCSI and other protocols, which are based on the TCP/IP protocol. iSCSI requires for each communication to build up a socket between an iSCSI initiator and an iSCSI target. If the iSCSI- or TCP/IP-stack inside the client can not be adapted to the demands of the storage cluster, it is not possible to transparently move the target endpoint of the socket connection without the use of an intelligent intermediate switch or server [15].

3.1 Analyzing the Expected Scalability

In a first step, we will analyze the expected scalability of storage clusters. We will assume that the performance is restricted by the interconnection network between the peers and from the peers of the storage to the clients. This assumption is valid for sequential access patterns as well as for random access patterns on Flash RAM-based hard disks. Furthermore, we will assume that each client is always connected with exactly one peer and that the clients are evenly distributed about the peers.

Assuming a fixed connection between a client and one storage brick inside the storage cluster and a striped or randomized data distribution scheme, the probability that an access can not directly be served by the master peer of a client growth linearly with the size of the storage cluster. If the cluster contains n nodes, only $1/n$-th of the requests could be served directly from a master peer. The remaining requests have to be forwarded from the master peer to peers containing the correct information.

In the following we will analyze the impact of this behavior on the scalability of the storage cluster. The network bandwidth that can be delivered from a single node system to its clients will be denoted by b. In the optimal case, the network bandwidth x_n delivered from one storage brick in a storage cluster with n nodes is equal to b and the total bandwidth delivered by n peers is $B_{total} = n \cdot x_n = n \cdot b$. In a real environment, we expect a behavior of type

$$B = f(n) \cdot \alpha \cdot b \tag{1}$$

as first order approximation, where $f(n)$ is a function expressing the scalability depending on the number of nodes n and α denotes a constant parallelization overhead (nearly) independent of n. In the investigated case, the parallelization overhead α is e.g. induced by a constant number of additional communication rounds between the peers exchanging requests.

Inside this extended abstract, we will consider the scalability of m-out-of-n codes. These codes have the advantage that the required redundancy to store data can become much smaller than for pure data replication; e.g. parity RAID with 4 data blocks and one parity block has an overhead of only 25%, compared with an overhead of 100% for mirroring with the same degree of data protection. This increase in storage efficiency normally includes a decrease in performance. The change of a sub-block requires that at least two sub-blocks are being read and two sub-blocks are being written to keep the parity block consistent. In the following, we will denote (due to consistency reasons inside this paper) the parameter n of the code as q and the parameter m as w.

We will show that this increase in storage efficiency can also be used to decrease network load if the environment is able to write full stripes. In this case, the redundancy blocks can be calculated without reading data from storage and without straining network bandwidth. It is possible to calculate the usable bandwidth for writing data for full-duplex connections as:

$$\begin{aligned} b &= \max\left(x_n + \frac{w}{q} \cdot \frac{n-1}{n} \cdot x_n, \frac{w}{q} \cdot \frac{n-1}{n} \cdot x_n\right) \\ \Rightarrow x_n &= \frac{q \cdot n}{(q+w) \cdot n - w} \cdot b \end{aligned} \tag{2}$$

where the first term of the $\max$-function depicts incoming communication and the second term outgoing communication from a storage node. Therefore, the overall bandwidth scales according to

$$B_{total} = \frac{q \cdot n^2 \cdot \alpha \cdot b}{(q+w) \cdot n - w} \approx \frac{q \cdot n \cdot \alpha \cdot b}{(q+w)} \tag{3}$$

If q is equal to w, the term describes full-duplex writes without replication. If $w = k \cdot q$, the scaling becomes equivalent to the scaling of a k-fold replication scheme.

4 Measurements

The aim of our measurements is to experimentally evaluate the influence of interconnection technologies on the scalability of a storage cluster which is using direct attached storage devices. The measurements have been performed on a Linux computing cluster, so it has been possible to scale up to a large number of storage bricks.

To outline the influence of interconnects on scalability, we have used internal RAM disks to be able to abstract from the influence of the used storage media. To overcome resulting caching effects inside the client computers, we have developed a virtual RAM disk that is able to consistently store a defined part of the RAM disk address space in memory and just returns random blocks for the rest. Therefore

it becomes possible to write a consistent boot block on an infinitely large RAM disk.

For each test, the adaption factor α has been calculated to minimize the average quadratic deviation from the expected results. If a test consist of p test runs for different numbers of nodes, x_i denotes the measured bandwidth for the i-th test run and ξ_i denotes the expected bandwidth for the same test, α is chosen in a way that it minimizes

$$\sum_{i=1}^{p} (x_i - \alpha \cdot \xi_i)^2 \quad (4)$$

4.1 Test Environment

To test the scalability of the storage cluster, up to 24 storage brick nodes and up to 24 clients, each equipped with two 1 GHz Pentium 3 CPUs and 512 MB RAM, have been used. Each of the storage brick nodes has exported a 1 TByte virtual RAM disk via iSCSI to all other nodes inside the storage cluster. Each server has been running RedHat Enterprise Linux AS 4 with a 2.6.9.42 kernel. The nodes have been connected by a 100 MBit/s Ethernet Cisco Catalyst 5509 switch that has been equipped with six WS-X5234-RJ45 24 node expansion modules. The backplane of the switch contains three busses, where each bus has a maximum throughput of 1.2 GBit/s. The maximum measured performance of each bus segment is 900 MBit/s that can only be observed for optimized communication patterns. Even if the cluster is based on elder technology, the resulting effects also apply to recent storage clusters.

For the scalability tests, each client has been connected to one node of the storage cluster. The physical (RAM) disks of the brick nodes have been grouped in one single storage pool. For each client node we have created two virtual volumes from the storage pool which have been exported to the client node via iSCSI. The data of each virtual volume has been scattered over all physical (RAM) disks of the storage pool. For iSCSI-target mode, we have used the iSCSI Enterprise Target version 0.4.12 driver. For iSCSI-initiator mode, we have used the iSCSI-initiator module that has been deployed with RedHat AS 4 and which is based on a Linux-iSCSI(sfnet)-driver.

IOmeter has been used as benchmarking environment. It consists of a set of agents for Linux, called dynamos, which are working as load generator on the client computers. The dynamos are managed by a server program on a Microsoft Windows PC. If not mentioned otherwise, the maximum number of outstanding IOs for each client has been set to 16, the access size has been set to 32 KByte, and the performance has been measured for 5 minutes for sequential writes after a ramp-up time of 30 seconds.

4.2 Measurement Results

Local Performance Many parallel solutions are able to scale performance in the number of nodes of the environment from 2 to n nodes, but have to cope with a significant parallelization overhead. This parallelization overhead often leads to a performance decrease for smaller environments compared to a local solution. Besides the examined overhead of the inter-node communication, this overhead can be induced in our case e.g. by the virtualization layer or network protocol stack.

In this section, measurement results for the performance of a local solution will be presented, where both client and server are on the same computer system. Furthermore we investigate the influence of the virtualization layer and the iSCSI-communication between one server and one client. In all cases, two workers are accessing two volumes.

In the first case, the dynamo agents directly access the virtual RAM disk on the same node. The maximum performance of the RAM disk is a sequential read throughput of 360 MByte/s for 32 KByte blocks and it can deliver up to 11,429 32 KByte random I/Os per second. The sequential write performance drops to 136 MByte/s. Both CPUs are under significant load. The situation changes slightly when a virtualization environment is put between the RAM disk and the IOMeter agent. The sequential write throughput drops to 120 MByte/s and the IO-rate drops by the factor 3/4. The reason is based on communication with the metadata appliance, which imposes additional delays for all first accesses to new regions.

In the next case, the server has been connected via iSCSI with a client computer. To directly measure the influence of the iSCSI communication between two computers, the server exports two virtual disk that only access the RAM disk. The throughput for sequential write accesses is 10.02 MByte/s for a 100 MBit/s Ethernet connection. The random I/O write performance is 313 I/Os per second or 9.78 MByte/s is nearly as fast as the sequential throughput. The last test measures the case when the RAM disk is exported as two virtual volumes. The sequential write performance drops slightly to 9,1 MByte/s, while the random I/O write performance decreases to 195 write I/Os. The decrease is based on the communication with the metadata server.

Scalability using RAID 1 The next test series is based on virtual RAM disks again and investigates the behavior of storage clusters applying data replication (additional test results, e.g. without data replication or for real disks are given in the full version of this paper). The RAM disks of all storage bricks are grouped inside a single storage pool. Each virtual disk derived from the storage pool has got a capacity of 40 GByte and two virtual volumes are combined to one mirror volume. Each client computer is again con-

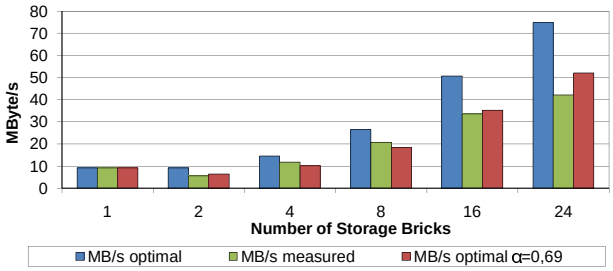


Figure 4. Scalability of mirrored RAM disks.

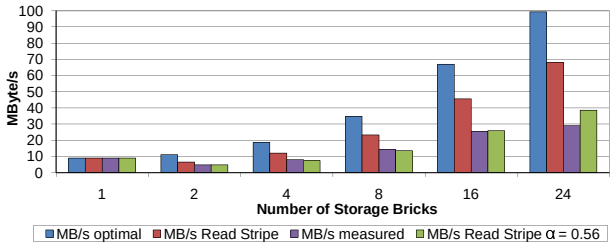


Figure 5. Scalability of RAID 5.

nected with exactly one cluster node and imports two mirror volumes. Therefore, 96 virtual volumes are set up for the 24 node test. Writes are send to both virtual volumes of a mirror volume, reads are performed by an arbitrary of both (see also [5]).

The measured performance for write tests only increases according to Equation 3 by a factor $1/3$. After scaling well from two nodes to 16 nodes, the performance for 24 storage nodes lacks behind the expected performance. This is based on the deployed switch. The measured performance of 42 MByte/s produces additional, internal traffic of 56 MByte/s between the three leaf boards, saturating the backplane with an overall traffic of 790 MBit/s. The expected performance of 52 MByte/s would already produce an overall traffic of 970 MBit/s on the backplane. Therefore, it is important to consider the internal traffic between the peers that can become much bigger than the external traffic. The factor α has been set to 0.69 to minimize the error, neglecting the test for 24 storage nodes[2].

Scalability of RAID 5 The theoretical assumptions leading to Equation 3 promise that the network load can be significantly reduced, compared to a k-replication of the data, by using m-out-of-n codes. Based on a single server write throughput of 9 MByte/s, this would theoretically lead to a throughput of up to 100 MByte/s for a 24 nodes 4-out-of-5 storage cluster, compared to a theoretical maximum throughput of 73 MByte/s for a 24 nodes cluster that mirrors data. Besides this expectations, the measured write performance lags behind the expected performance and even behind the measured performance for Mirroring.

The reason is the special handling of request inside the iSCSI target driver inside the storage nodes. Each 32 KByte write request is split into 4 KByte requests which are successively handled by the underlying page cache layer and block layer. Therefore, the first block of each stripe is handled as a new stripe and requires to fetch the remaining blocks of the stripe from the other peers to calculate the new parity block, leading to additional 32 KByte of read requests. Furthermore, 40 KByte of data have to be written to the peers. This leads to the following calculation for the used 4-out-of-5 code:

$$\begin{aligned} b &= \max\left(\frac{13\cdot n-9}{4\cdot n}\cdot x_n, \frac{9\cdot n-9}{4\cdot n}x_n\right) \\ \Rightarrow x_n &= \frac{4\cdot n}{13\cdot n-9}\cdot b \end{aligned} \tag{5}$$

and therefore the environment scales according to

$$B_{total} = \frac{4\cdot n^2\cdot\alpha\cdot b}{13\cdot n-9} \approx \frac{4\cdot n\cdot\alpha\cdot b}{13} \tag{6}$$

Using an adaption constant α of 0.56 leads to a nearly perfect approximation of the measured behavior. Again, the 24 node test requires too much communication to be able to scale according to the predicted results.

5 Conclusions

Storage clusters start to become an interesting alternative to standard storage architectures. Today, already a number of storage vendors is offering different storage cluster alternatives, starting from block based storage clusters up to scalable file servers. In this paper, we have shown that the hardware costs for a storage clusters are below or comparable to dedicated entry-level storage architectures and still an order below high-end storage systems.

As shown inside this paper, the interconnects can easily become the limiting performance factor for sequential accesses, especially if data has to be replicated. Nevertheless, storage clusters are really able to scale performance in the number of nodes. This performance increase is not only based on a larger number of spindles, but also on more communication interfaces and larger, aggregated caches. Summarized, storage clusters are able to scale performance and capacity while delivering a high degree of reliability and are able to overcome limitations imposed by centralized storage architectures.

[2]The used iSCSI configuration producing the test results has not been optimized and better results can be achieved which would lead to a higher α-value.

References

[1] M. Abd-El-Malek, W. Courtright, C. Cranor, et al. Ursa Minor: Versatile Cluster-based Storage. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2005.

[2] P. T. Breuer, A. M. Lopez, and A. G. Ares. The Network Block Device. *Linux Journal*, 73, 2000.

[3] A. Brinkmann and S. Effert. Cost effectiveness of storage grids and storage clusters. In *15th Euromicro Conference on Parallel, Distributed and Network based Processing*, pages 517–525, Naples, Italy, 2007.

[4] A. Brinkmann and S. Effert. Inter-node communication in peer-to-peer storage clusters. In *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies (MSST)*, pages 257–262, San Diego, California, 2007.

[5] A. Brinkmann, S. Effert, M. Heidebuer, and M. Vodisek. Distributed MD. In *Proceedings of the 3rd International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, 2005.

[6] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, Adaptive Placement Schemes for Non-Uniform Distribution Requirements. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.

[7] T. Clark. *Designing Storage Area Networks*. Addison-Wesley, 2nd edition, 2003.

[8] Equallogic. PS Series - Intelligent iSCSI Storage Arrays. Product Brochure, 2005.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[10] R. J. Honicky and E. L. Miller. Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution. In *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[11] A. Huffman and J. Clark. Native Command Queuing (NCQ). Joint White Paper of Intel and Seagate, 2003.

[12] D. Karger, E. Lehman, T. Leighton, et al. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, 1997.

[13] E. K. Lee and C. A. Thekkath. Petal: Distributed Virtual Disks. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1996.

[14] H. Mason, S. Peiffer, and H. Smith. SAS and SATA team up for enterprise. *SNS Europe*, 6(3):28–30, 2006.

[15] V. Olaru and W. Tichy. On the Design and Performance of Kernel-level TCP Connection Endpoint Migration in Cluster-Based Servers. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2005.

[16] K. W. Preslan, A. P. Barry, J. Brassow, et al. Implementing Journaling in a Linux Shared Disk File System. In *Proceedings of the 17th IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2000.

[17] A. Rajasekar, M. Wan, R. Moore, and T. Guptil. Data Grids, Collections and Grid Bricks. In *Proceedings of the 20th IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2003.

[18] Rancho SysTech Inc. RTSASR-12X 12-Port Serial Attached SCSI (SAS) Edge Expander. White Paper, June 2005.

[19] Y. Saito, S. Frølund, A. C. Veitch, A. Merchant, and S. Spence. FAB: building distributed enterprise disk arrays from commodity components. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004.

[20] F. B. Schmuck and R. L. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST)*, 2002.

[21] Seagate Technologies. SCSI Inflection Point: The New Era of Serial Attached SCSI. White Paper TP-528, June 2004.

[22] F. Tomonori and O. Masanori. Analysis of iSCSI Target Software. In *Proceedings of the 2nd International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, 2004.

[23] W. Wang, H. Yeo, Y. Zhu, and T. Chong. Design and development of Ethernet-based storage area network protocol. In *Proceedings of the 12th IEEE International Conference on Networks (ICON)*, 2004.