THE POSSIBILITIES ARE INFINITE FUJITSU

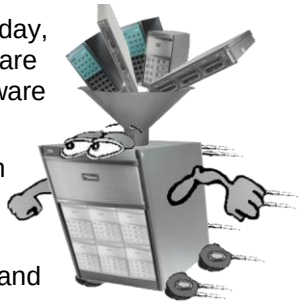
WHITE PAPER

Version 1.1
March 2008

Performance Report XEN (SLES 10)

Pages 31

Abstract



Server virtualization is at present one of the major topics of the IT environment. Today, modern hardware is often so powerful that it is not run to full capacity by a single software application. Virtualization is ideal for old software solutions, particularly where old hardware is replaced.

In addition to established solutions from VMware und Microsoft, XEN, an »Open Source« implementation, is regarded as sufficiently mature for productive deployment and has become an integral part of the most important Linux distributions.

This document looks at the use of XEN on Novell SUSE Linux Enterprise 10 SP1 and attempts to make recommendations for practical use and to forecast performance data.

Content

Introduction

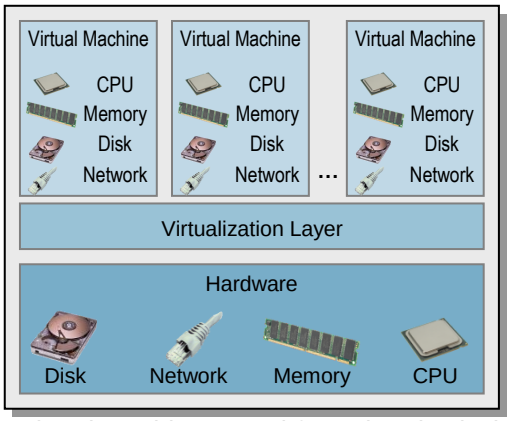
The subject of server consolidation is one of the principal topics when it comes to saving costs in the IT environment. A better and more effective server workload as well as a reduction in the number of servers are called for. Nowadays individual applications are allocated to a dedicated server so as to prevent any reciprocal interference. Therefore, with a large number of applications the number of servers in a data center also increases. As a result of this allocation the available computing performance frequently lies idle because the individual applications do not utilize the server to the full.

Using virtualization enables several virtual servers to be consolidated on a single physical server. Thus the use of virtual machines even enables several different operating systems, e.g. various Linux derivates or Windows versions, to be run in parallel on the same physical server.

A further advantage of virtual servers is the possibility of operating legacy systems or existing applications together with their environments in virtual machines.

Currently available on the market are various popular virtualization products for Intel-based servers, this document concentrates primarily on XEN 3.0.4 with SUSE Enterprise Linux 10 SP1 as the host operating system.

What is a virtual server?



Virtualization is a technology that allows several operating systems to be run on a single physical server at the same time. Virtualization can be realized with the help of hardware or software. In the case of software-based virtualization a virtualization program is used to insert an additional layer, the so-called virtualization layer, on the physical server between the actual system resources and the virtual servers, also known as virtual machines. The hardware of the physical server is made available to the virtual servers in a suitable form via the interface of the virtualization layer. In this way, the virtual machines can be fully separated and isolated from each other.

Hardware resources - emphasis here is always placed on the four core components CPU, memory, network and disk resources - are mapped in every virtual machine. Each access of a virtual machine to and from the physical hardware of the host server passes through the virtualization layer.

As a means of distinguishing the servers, the physical server is also denoted as the »host« and the virtualization layer as the hypervisor (and also as »Virtual Machine Monitor«/VMM), on which the virtual servers, also known as virtual machines (VM), run with their guest operating systems. The virtualization layer completely separates the virtual machines from the host hardware and its hardware/driver dependencies. In such a configuration it is possible, for example, to run a virtual server which has been created on a PRIMERGY RX300 S4, on a PRIMERGY RX600 S4.

The volume of hardware resources of the virtual machine can frequently be changed manually using the virtualization program. In this way, it is possible to change CPU resources during ongoing operations. Depending on the requirements of the virtual machines they can be allocated more or less CPU time.

Each virtual machine must be seen as a separate server, which can in turn be run on the host fully independently of the other virtual machines. The virtual machines are isolated from each other to the effect that data security is ensured even with business-critical and confidential data.

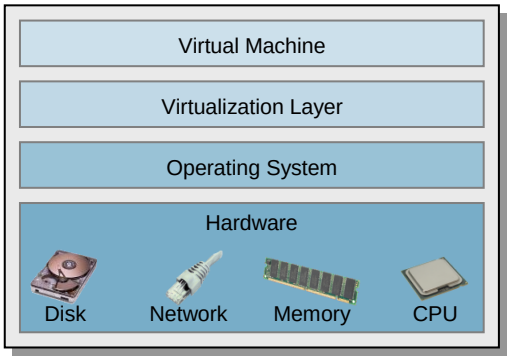
Virtual machines (VMs) can in the simplest case consist only of a configuration file, a disk file and a log file, which makes it relatively easy for the administrator to back them up.

Typical virtualization architectures

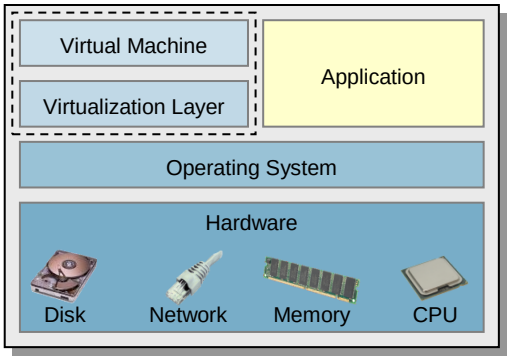
In principle, there are four types of virtualization techniques, which enable one or more virtual servers to be run on a shared, physical hardware platform.

Type 1:

In the first variant an operating system is installed on a physical server, upon which a virtualization program is positioned. Consequently, a virtualization layer is added to the operating system, on top of which the virtual machine is positioned. Every CPU, disk, memory or network access must pass through this layer. The diagram on the right shows the basic structure.



The advantage of this type of virtualization is that in addition to the virtual server other applications can be run on the host operating system, as shown in the figure on the left.

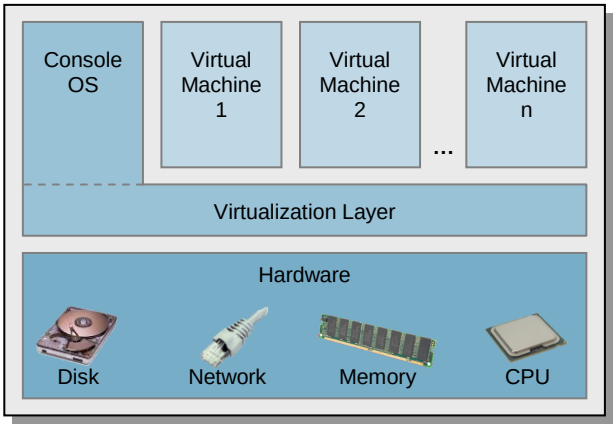


In this way, applications can be run on a physical server in parallel with the virtual server. The disadvantage of such a virtualization solution is the actual overhead of the host operating system. The host server runs system services, which need resources that are only required to operate the applications, and not to operate the virtual machines. The performance of the virtual machines is consequently reduced.

All hardware accesses to and from the virtual machine must pass through the virtualization layer and the host operating system.

Type 2:

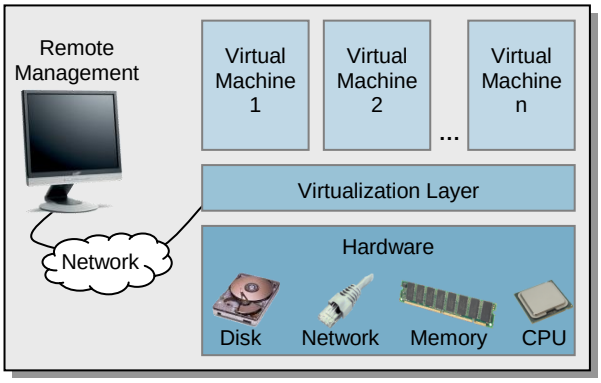
The second variant actually does not use a host operating system, as is the case with virtualization type 1, but already implements all the functions directly required for virtualization in the virtualization layer. Since this also includes control of the I/O devices, the virtualization layer must also implement the necessary drivers or at least provide a general interface, via which third-party drivers can be integrated.



A special auxiliary operating system (»Console OS«), the scope of which has been reduced to the functionality genuinely required, is used for the administration of the system. The auxiliary operating system itself can already be seen as a VM with a special status, thus it can have e.g. very direct access to the I/O devices.

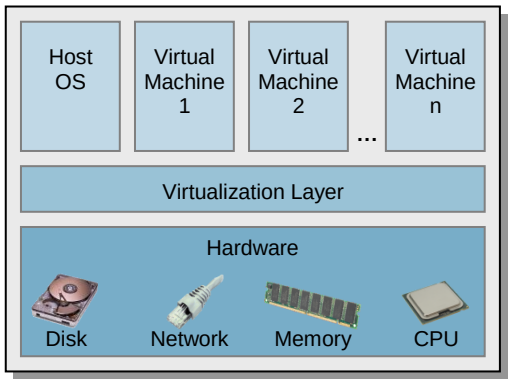
The advantage of such a virtualization solution is that it is not burdened with the overhead of a host operating system, and hardware access to and from the virtual machine only has to pass through one layer, the virtualization layer. For example, the VMware ESX Server is based on this concept and uses a specially adapted Linux version both as the basis for the virtualization layer and for the auxiliary operating system.

Type 3:



The third variant is the most straightforward implementation. In principle it is very similar to the second variant, in that it also provides all the functions directly required for virtualization, including I/O control, itself. In contrast to the second variant, however, merely administration interfaces and no administration functions/programs are provided directly on the system. As a result, the console operating system is no longer applicable, as is the resource consumption it causes. The administration functionalities also required with this variant must be provided by an external system. Virtualization solutions that are based on this principle can be so compact that they can in fact already be provided by the firmware of a system. One example worth mentioning for such a solution would be the VMware ESX Server 3i.

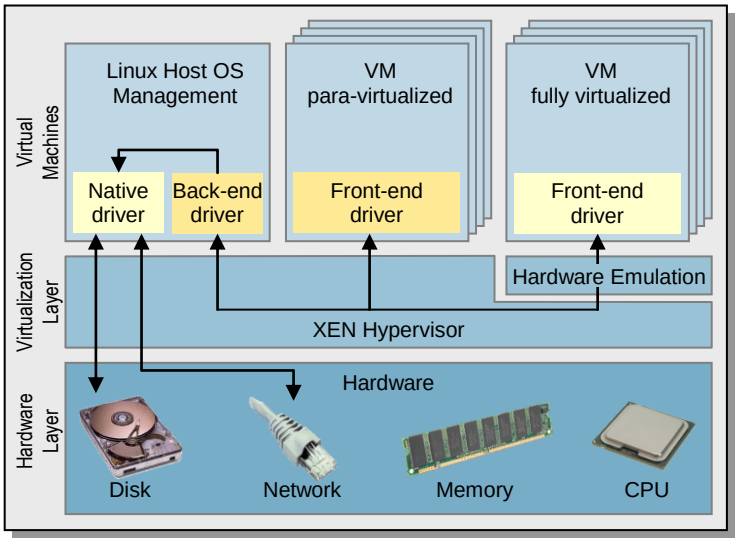
Type 4:



The fourth variant refers to both variant 1 and variant 2. As with variant 1, a fully fledged host operating system is used to perform all hardware accesses. However, this host operating system already runs under the supervision of the hypervisor (refer to variant 2), but compared with conventional VMs it has the privilege of being able to access the hardware directly. The I/O operations of the conventional VMs are not performed by the hypervisor, but by this host operating system. The advantage of this approach can be seen in the fact that the hypervisor can not only be implemented in a compact way here, but has, as in the second variant, full control over the host system and can in this way - at least with regard to CPU and memory resources - avoid the overhead caused by a regular host operating system. However, with regard to I/O activities the overhead not only continues to exist, but can - depending on the architecture of the host operating system - be even higher than in variant 1. In order to avoid this overhead hypervisors, such as XEN, provide the possibility for individual HW controllers, such as a PCI card, to be allocated to an individual VM on a dedicated basis. Consequently, very fast access from the VM is possible, but this is paid for with the disadvantage that only one VM can access this HW at the same time. Moreover, HW independence is no longer given, therefore such a VM can no longer easily »move« to another HW platform.

Since the host operating system analog to variant 1 can be a fully fledged operating system, it is in principle here also possible to provide further server services in addition to the actual virtualization functionality.

General overview of XEN



XEN is virtualization software that has in the meantime become a part of many Linux distributions. Although XEN, similar e.g. to the VMware server, can be regarded as an add-on to a standard operating system, it nevertheless implements an independent virtualization form of type 4. The decisive difference to a virtualization solution of type 1 can be seen in the fact that XEN in principle also virtualizes the host operating system and thus actually degrades the latter to an auxiliary operating system. However, compared with a regular VM, the VM of the host operating system has a privileged status, which allows it e.g. to access the hardware directly. This is necessary because the privileged VM is used by the conventional VMs as a kind of »proxy« for their I/O operations. The device drivers in the VMs are denoted in the diagram as front-end drivers. The I/O requests are sent by the front-end drivers to the back-end drivers in the privileged VM, which pass the requests to their own native drivers.

All the VMs are denoted in XEN as »Domain« or »Dom« for short and are numbered consecutively beginning with 0. The privileged status of the VM of the host operating system is also expressed in the naming convention for this VM. The latter defines »Dom0« as the name for the VM of the host operating system, whereas all other VMs are denoted with the generic term »DomU«. Since practically each virtualization solution defines its own terms, the far more conventional abbreviation »VM« is used in this document instead of »DomU«. On the other hand, the name »Dom0« is also used below, because none of the other virtualization solutions stated here has an appropriate equivalent.

In comparison with other virtualization solutions, the functional scope of the XEN hypervisor is reduced to controlling CPU and memory resources as well as handling asynchronous events (e.g. interrupts). It also controls the scheduling of the VMs, without influencing the scheduling within the VMs.

Virtualization forms

Due to the history of their development in the pre-virtualization era the classic x86/x64 processors have various characteristics (particularly in the privileges for CPU instructions and in memory management), which make virtualization considerably more difficult. Today's virtualization solutions have to avoid these deficits in a complex way with software. Several virtualization concepts exist for this purpose.

Para-virtualization is a form of virtualization, in which the operating system of the VM »knows« that it is virtualized. It supports virtualization by only using the CPU in such a way that it can be virtualized without any problems. This calls for modifications in the operating system kernel and in the device drivers. Virtualization support through the processor (Intel-VT [L4] / AMD-V [L5]) is not necessary in this form of virtualization and does not entail any advantage. Para-virtualization is at present the form of virtualization which has by far the lowest overhead and thus also the smallest losses in performance compared with a native operating system. Accordingly, the Dom0 is based on this form of virtualization in XEN, because it must perform all the I/O operations in an acting capacity for the DomU VMs.

With so-called **full virtualization** the OS of the VM can remain unchanged. The problem mentioned at the outset is resolved by newer x86/x64 processors with extensions (Intel-VT [L4] / AMD-V [L5]) that can be used by the hypervisor. Since no OS modifications are necessary with this form of virtualization, non-adapted operating systems can also be virtualized with it. The primary disadvantage of full virtualization can be seen in the fact that it entails on the one hand greater losses in performance than para-virtualization and that implementation is very complex despite the innovative processor support. The background to this among other things is the fact that with classic full virtualization the most important hardware components of a typical computer have to be elaborately emulated in software (»Hardware Emulation« in the diagram on the previous page). Generally, losses in performance cannot be avoided here, but can mostly be minimized through additional »driver packs«. Through this add-on special, in principle para-virtualized disk and LAN drivers are also made available to a fully virtualized VM in order to at least achieve a similar performance for low-level activities, as in a para-virtualized VM.

XEN originally only offered para-virtualization, but full virtualization has also been possible since version 3. As all the previously described XEN virtualization variants are considered more closely with regard to their performance in this document, the terms listed below are used in the rest of the document for a more accurate specification of the form of virtualization:

VM	Specifies a virtual machine in a quite general way, the actual virtualization form does not play a role here.
‘Full’ VM	Denotes a fully virtualized VM.
‘FullEx’ VM	Specifies a fully virtualized VM, extended to include the driver pack from Novell as an add-on for SLES10 [L7] for network and disk IO.
‘Para’ VM	Denotes a para-virtualized VM.

I/O Structure

A detailed description of the way in which I/O operations run under XEN would fall outside the scope of this document. Therefore, the basic process is only roughly explained here.

The fundamental I/O concept of XEN is based on the fact that either the Dom0 or a ‘Para’ VM configured as a driver domain performs - in an acting capacity for the conventional VMs - the I/O operations that they initiated. The SW components that control this procedure are denoted in current literature as »front-end device drivers« (initiating VM) and as »back-end device drivers« (Dom0 and driver domain). Ideally, communication between the two driver instances takes place via »shared memory« and semaphores in order to minimize in this way the actual communication overhead. This is at least possible with ’Para’ VMs and, with a certain additional overhead, also with ‘FullEx‘ VMs. Regardless of how quickly actual communication is handled, with this concept there will always be the problem that two interdependent instances here can really only work in parallel in optimal situations - even on multiprocessor systems. Therefore, in addition to the pure communication costs, latency times arise, which result from the scheduling of the two instances. A virtualization solution of type 1 or 2 could work in an undoubtedly more straightforward way in this regard, because it would always be possible here to trigger the execution of an I/O request immediately after its receipt. Prerequisite to this is that the used host operating system provides an asynchronously working I/O API.

With a ‘Full’ VM communication between the back-end and front-end driver is considerably more elaborate, because there is no direct communication at all in the actual sense. The front-end driver believes it is accessing real hardware and behaves accordingly. The back-end driver is compelled to monitor these activities, interpret them logically and perform them in an acting capacity. This is complex and accordingly has a decidedly negative impact on performance.

Not only does communication between the front-end and back-end driver have a decisive influence on performance, so does the way in which the Dom0 and the relevant driver domain I/O operations are handled.

XEN on the basis of Novell SLES 10

Novell SUSE Linux Enterprise Server 10 SP1 (hereinafter referred to as SLES10) includes XEN version 3.0.4 with additional modifications from Novell. Server systems based on x86 architectures are supported both in a 32-bit and a 64-bit version. The following applies for guest operating systems: 64-bit operating systems can also only be virtualized on 64-bit hardware. It is possible to run both para-virtualized and fully virtualized VMs on the same host system; however, full virtualization necessitates a processor with Intel-VT or AMD-V extensions. Although the XEN integrated in SLES10 could actually virtualize each guest system supported by the XEN community variant, Novell nevertheless limits the list of supported operating systems. Since the list of supported guest operating systems is subject to certain dynamics, we refer in this respect to the Novell web page http://www.novell.com/products/server/virtualization.html [L6].

As a distinguishing feature compared with the competition, Novell offers a chargeable, additional package entitled »SUSE Linux Enterprise Virtual Machine Driver Pack«, which also provides para-virtualized disk and network card drivers for a number of operating systems as part of full virtualization. The »driver pack« is an add-on that can be obtained from the Novell web site http://www.novell.com/products/vmdriverpack [L7].

Practical use

The measurements on which this document is based were not made in accordance with the original SP1 status. The SLES10 kernel had to be updated to version 2.6.16.53-0.16 and XEN to version 3.0.4_13138-0.52, as otherwise the measurements would have been impossible on account of various problems. The patches are automatically installed for a customer when he enables the YaST2 online update from the Novell server by entering the registration code. Alternatively, the patches can be manually downloaded and installed (http://support.novell.com/linux/psdb [L8]).

As the standard measuring system for the virtualization host a PRIMERGY RX300 S3 (2 × Intel Xeon Quad-Core, 5365, 3 GHz, 16 GB RAM) was used for this document.

Performance analyses were also performed with the driver pack for Windows 2003 (version 1.1.3-6), as performance can be distinctly improved as a result. However, with Windows 2003 SP2 a number of problems exist in the VMs as a result of this driver pack, but which can be avoided:

- If the VM configuration includes a virtual disk (VHD) declared as »hdd«, the consequence of this is an endless boot process. Thus, a VHD should never be declared as »hdd«. One consequence of this is that in a configuration with three VHDs and a virtual DVD drive the DVD drive would - contrary to the normal »hdc« standard - have to be declared as »hdd«. However, on account of the Novell driver pack the declaration can also be made as »hde«, as this lifts the limitation of at most four virtual disks/DVDs per VM, which normally applies for fully virtualized VMs without a driver pack (which Novell unfortunately does not describe in the driver pack documentation).

- While the operating system in the VM without an installed driver pack allocates ID 0 to the boot HD, the latter with an installed driver pack is issued ID 2. All other HDs subsequently follow suit. This is not a real problem, more of a minor flaw. However, in this way it is initially not possible with programs such as *Iometer* (see the following section Measurement methods), which expect end-to-end numbering of the HDs beginning with zero, to access a raw disk at will. This is only possible once all HDs, except for the boot HD, have been deinstalled in the Windows Device Manager. As part of the subsequently performed hardware rescan the »re-found« HDs are consecutively numbered beginning with zero. Consequently, access with *Iometer* is now possible.

- One effect of the driver pack that arises when a bootable CD or DVD is in the configured DVD drive during the booting of a VM is more serious. In such a situation, and depending on the declaration of the virtual DVD drive, the disk driver generates one or even two »imaginary HDs« (IHD). Only one single IHD is generated with ID 4 for a DVD drive declared as »hdc«. If the declaration is »hdd«, two IHDs are generated with the IDs 0 and 5. In both versions the IHDs force themselves between the genuinely configured VHDs and thus change their numbering. IHD generation is not just a minor flaw, it leads to problems in disk management programs. This is due to the fact that under Windows partitions can only be created on HDs on which Windows has already written a unique signature. However, such an attempt fails with IHDs. Unfortunately, some programs attempt to repeat this process with every program start, which results in considerable waiting.

With the OS version used the following problem occurs in the Dom0:

- The »xentop« command reports dubious values in the CPU load of the Dom0 as a result of the I/O activities. These values do not correlate with the values of commands, such as »mpstat«, namely to the extent that the xentop values are in comparison considerably excessive. No decisive clarification could be found as to which command is supplying authentic values. Analyses, which were performed to clarify the problems by means of network load between the SLES10 system and a comparable Windows counterpart, resulted in values for the Windows system that are in fact close to the »mpstat« values. However, on account of the quite different architecture of both operating systems this can only be an indication. As a consequence of this problem it is not possible in this document to provide any details about the effective system load during a performance measurement.

Measurement methods

This section is to present the measurement methods for performance analysis, the measuring tools used and the measurement environments. The respective configuration of the server and storage hardware used, as well as the configuration of the native and virtual operating systems are described specific to the measurement method.

Benchmarks

Since there is no universal tool for the analysis of a complex entity such as a virtual server, various benchmark tools are used depending on the purpose.

Iometer

Iometer [L9] is an Open-Source measuring tool that is excellently suited for the generation of disk and network load on a rather lower system level. Version 2006.07.27 is used. In the Windows environment the original download compilation is used, in the Linux environment on the other hand a modified version is used. This is due to an error in the query to determine if errors have occurred in the program run and which frequently prevents a correct end to a measurement run, especially with multiprocessor configurations. The modifications affect the file *IOCompletionQ.cpp*; where line 308 must be changed as follows:

*if ((cqid->element_list[i].error == 32) || (cqid->element_list[i].error == 104) || (DWORD) * bytes_transferred < (DWORD) 0) {*

This is only a »quick and dirty« solution which has proved to be adequate for regular measurement operations.

vServCon

For measuring server consolidation in virtual environments Fujitsu Technology Solutions has developed the benchmark »vServCon« [L11], which is based on »vConsolidate« [L10] from Intel. »vServCon« comprises several standard benchmarks. Each of the standard benchmarks is allocated to a dedicated virtual machine (VM). These VMs then form a »tile«. Depending on the performance capability of the underlying server hardware, you may as part of a measurement also have to start several identical tiles in parallel in order to achieve a maximum load. A detailed description of this environment can be found in the document »vServCon - Benchmark Overview« [L11].

SPECjbb2005

The SPECjbb2005 benchmark [L12] is a JAVA-based benchmark and measures the performance of server-side Java through the emulation of a 3-tier client/server system with the focus placed on the middle tier. See the document »SPECjbb2005 - Benchmark Overview« [L13] for a detailed description of this benchmark. For use as part of the benchmark framework vServCon the benchmark was modified according to the vServCon specifications. This includes cyclical sleep pauses, as otherwise the VM would on account of the absolute lack of I/O accesses with this benchmark use its full CPU time as allocated by the hypervisor and would thus show a rather untypical load profile for server applications in a VM. Therefore, no compatible result is generated, but merely a single indicator that provides information about the number of transactions made.

The benchmark runs directly on the system and does not need any external load generators.

SysBench

Sysbench [L14] is an »Open Source« benchmark for databases that is available for a large number of different target platforms.

SysBench is used as part of the vServCon framework. A version modified by Intel is also used here, and is based on Sysbench version 0.3.3.

The benchmark runs directly on the system and does not need any load generators.

WebBench

The benchmark WebBench 5.0 [L15] is a benchmark used to determine the performance of a web server in a client/server environment. Client PCs are used here to simulate web browsers, which send requests to the web server and log performance-relevant access information after receipt of the data.

Measurement environment

Depending on the benchmark used it is necessary to define a suitable VM. If a native system is measured as a comparison, an identical CPU/RAM configuration must be established by means of appropriate parameters in the configuration of the grub-boot loader and »boot.ini« respectively.

Iometer

Number of CPUs	1 core
Available RAM	1536 MB
Disk subsystem	FibreCAT CX500 RAID0 made up of five 36 GB hard disks with 15,000 rpm
Operating system	Microsoft Windows 2003 R2 Enterprise x64 Edition (SP2)

SPECjbb

Number of CPUs	2 cores (if possible)
Available RAM	2 GB
Disk subsystem	FibreCAT CX500 RAID0 made up of five 36 GB hard disks with 15,000 rpm
Operating system	Microsoft Windows 2003 R2 Enterprise x64 Edition (SP2) SLES10 SP1 64-bit (preferred)
Application	BEA JRockit R27.2.0

SysBench

Number of CPUs	2 cores (if possible)
Available RAM	1536 MB
Disk subsystem	FibreCAT CX500 RAID0 made up of five 36 GB hard disks with 15,000 rpm
Operating system	Microsoft Windows 2003 R2 Enterprise x64 Edition (SP2)
Application	Microsoft SQL Server 2005

WebBench

Number of CPUs	1 core
Available RAM	1536 MB
Disk subsystem	FibreCAT CX500 RAID0 made up of five 36 GB hard disks with 15,000 rpm
Operating system	SLES10 SP1 64-bit
Application	Apache 2

Performance Analyses

Every access to and from the virtual machine must pass through the virtualization layer. The task of this layer is to interpret and convert the I/O operations. All processor, memory, disk and network accesses are from the host's viewpoint converted for the virtual machine. Conversely, all accesses from the virtual machine to the host must also be converted.

This »transformation« costs computing performance and thus also computing time. The following performance analyses are intended to provide information about the extent to which this virtualization under XEN/SLES10 affects the performance of the virtual machines.

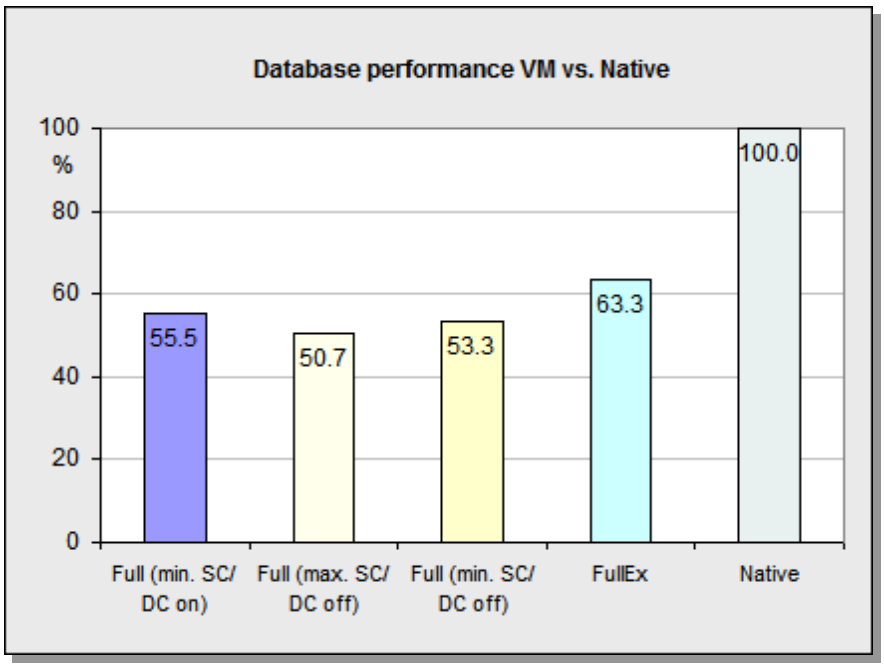
Below the four performance-relevant components

CPU **Memory** **Disk** **Network**

are first looked at individually, although CPU and memory are so interwoven that they are considered jointly.

CPU and Memory

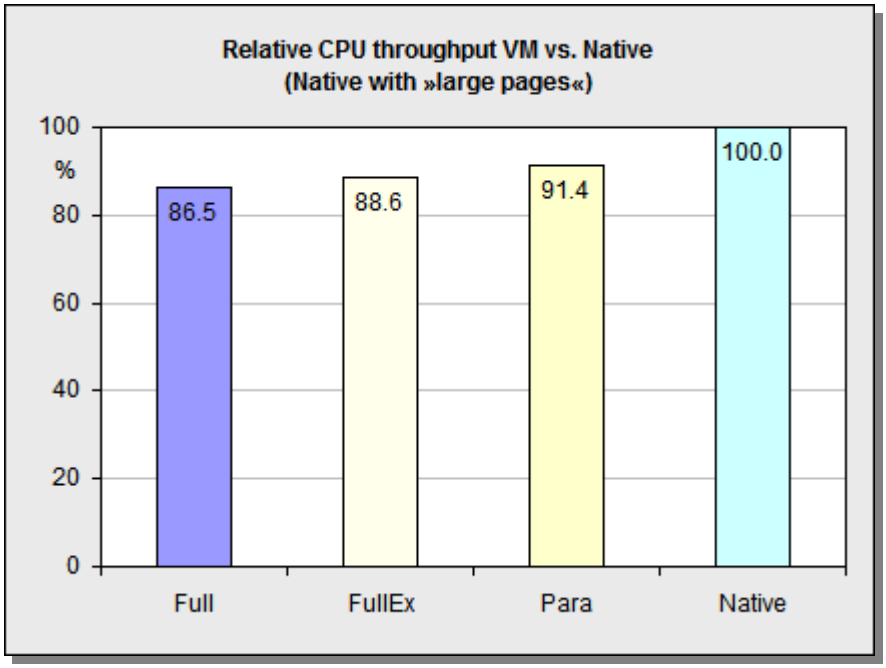
The diagram opposite depicts the performance ratio between the different virtualization variants. The para-virtualized VM shows an excellent performance here with only 1.2% loss in performance compared with a native system. With the fully virtualized VMs performance sinks by up to 6.5%.



The differences in performance between the individual forms of virtualization stem from memory management. From the view of the hypervisor, memory management for a para-virtualized VM is considerably easier and thus quicker to handle than for a fully virtualized VM. Furthermore, the kernel mode share also has a performance-reducing influence in the fully virtualized VMs, because despite the use of the special virtualization functions of the processors the execution of code in the kernel mode represents considerably more outlay in full virtualization for the hypervisor than when implemented in user mode. Depending on the kernel mode share of an application it is also possible for significantly larger deviations to occur. The influence of the kernel mode shares can also be clearly seen in the comparison of the 'FullEx' VM and the 'Full' VM. The 'FullEx' VM, which only differs from the 'Full' VM through the use of the para-virtualized disk and LAN drivers, performs 2.3% better, because the kernel mode shares are reduced as a result of the special disk and LAN drivers.

For memory-intensive applications, however, a larger discrepancy may result between the performance of a native system and the VM if they use the operating system option »large pages«, which by means of a single page table entry enables large contiguous parts of the physical memory to be addressed. As regards memory fragmenting, the standard »small pages« is usually the appropriate choice. However, for memory-intensive applications, such as Oracle Database, the option »large pages« is an important means of optimization. Unfortunately, however, XEN does not support any »large pages«, it merely offers an emulation. An application that profits from »large pages« will therefore show a considerably larger loss in performance in a VM, as is illustrated in the diagram below.

The CPU/memory analyses were performed with a modified version of the SPECjbb2005 benchmark. Both the native system and the VMs each consisted of a CPU core, 2 GB RAM and were run with SLES10 SP1 64-bit. In the case of the native system an identical CPU/RAM configuration was established by means of appropriate parameters in the configuration of the grub-boot loader.



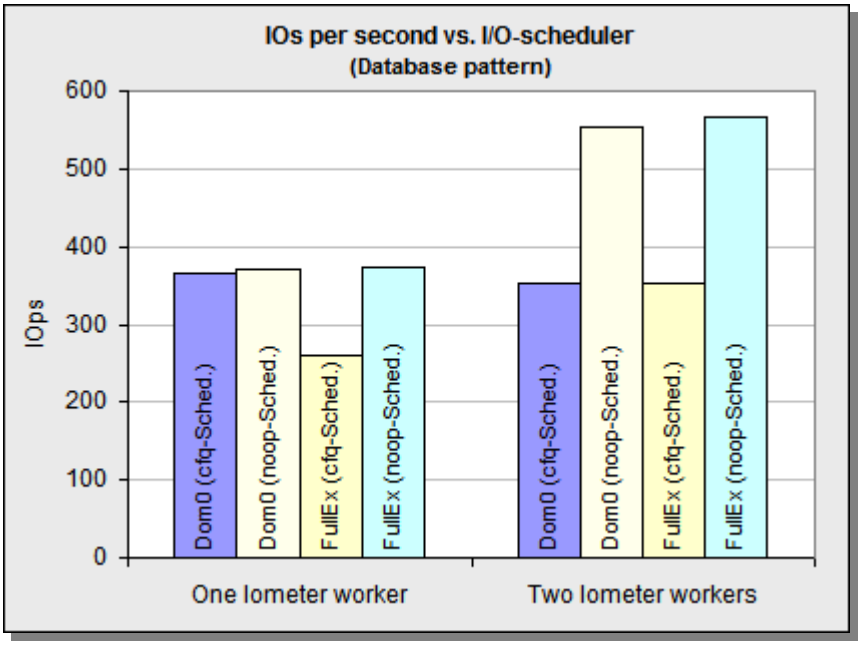
Disk I/O

Disk I/O is a complex structure. Not only are there various access patterns for disk I/O from sequential to random access, also with different block sizes and different parallelism, but also because the individual forms of virtualization implement different strategies in the handling of caches and in intervention in the I/O flow. Therefore, it is necessary to firstly deal with these features of the implementation under XEN prior to the analysis of disk-I/O behavior.

I/O Scheduler

Eminent importance is attached to the I/O schedulers in the disk-I/O structure of Linux, every I/O operation passes through them. The task of the I/O schedulers is to »optimize« the processing of I/O operations. What happens during this optimization depends on the respective I/O scheduler; in this way it is e.g. possible to put several I/O operations together to form a single one. Intensive use is made of this feature in connection with the system cache. Individual I/O schedulers also perform a re-sorting of the I/O operations based on the sectors addressed on the disk so as to minimize any unnecessary movements of the write/read heads. Here we are dealing with a functionality that not only provides powerful storage systems, but also modern SATA HDs (Native Command Queueing). Since it is hardly possible to optimally support all load scenarios with one individual I/O scheduler design, Linux makes a total of four different I/O schedulers available. The fact that the I/O schedulers can be individually set for every disk is an important feature especially for the connection of storage systems, because their internal attempts at optimization capable of being impeded through the optimization of the I/O schedulers. Under this aspect the »cfq« scheduler and the »noop« scheduler were looked at more closely as part of this document.

The »noop« scheduler is the simplest of the four I/O schedulers. Except for consolidating the individual I/O operations, it implements no further optimization. The »cfq« scheduler is in this regard clearly more complex, for example it is in a position to sort I/O operations on the basis of sector addresses. The »cfq« scheduler is the standard I/O scheduler under SLES10. The diagram shows that in a configuration with only one Iometer worker (that is a load-generating thread) the I/O schedulers in the Dom0 have no great influence on performance. Unlike 'FullEx' VM, here the »noop« scheduler achieves decidedly better results.



If the measurement structure is modified to the effect that two Iometer workers generate the same load in parallel on the same physical disk, the better performance of the »noop« scheduler can also be seen in the Dom0. The »noop« scheduler is in a position to significantly increase throughput both in the Dom0 and in the VM. On the other hand, with the »cfq« scheduler the overall throughput of the Dom0 sinks on account of the second Iometer worker - even below the throughput that was previously achieved by a single Iometer worker. However, the VM can on account of the second Iometer worker at least draw level with the Dom0 and is thus somewhat better than before with only one Iometer worker.

The two series of measurements show that at least in connection with a powerful storage system the »optimizations« of the I/O schedulers seriously influence performance. The simpler »optimizations« of the »noop« scheduler consequently have a decidedly better performance here. In this regard, it is advisable to always use the »noop« scheduler and particularly with Linux-based VMs to avoid optimizations in a second place through its internal I/O schedulers.

When examining the I/O schedulers one discrepancy occurred in SLES10. Although the »cfq« scheduler should be enabled as standard, with it the throughput of the »noop« scheduler is always achieved directly after booting. Only after a different I/O scheduler has been set or the »cfq« scheduler, which has actually already been set, has been overwritten, does it show typical throughput values. In this respect, doubt must be expressed as to whether the »cfq« scheduler indicated is really the standard scheduler.

System cache of the Dom0

The system cache buffers both read and write accesses and can use the entire free memory of the Dom0 for this purpose. Tests with »iostat« suggest that write for the modified cache pages is only effected on a time-dependent basis to a limited extent and primarily through displacement. Thus, for example in an Iometer measurement within a VM, it could be seen that according to »iostat« virtual disk areas written by Iometer were also actually written onto the physical disk only one hour after the measurement. This cache behavior is not transparent for the VMs and with regard to data integrity is even unwanted if the host were to have an emergency power supply (in the event of a Dom0 crash the data written in the VM by the application has not yet been passed on to the physical disk). Generally, the cache is more counterproductive than useful during virtualization, because the VMs also implement a cache at least on the level of their operating system and thus two independent instances possibly have the same data in their cache. This represents an additional overhead and accordingly results in losses in performance.

Both para-virtualized VMs and fully virtualized VMs with the Novell driver pack are affected by the Dom0 system cache when their virtual HDs are mapped onto a file within the host file system. With other configurations (e.g. a virtual HD on one partition) they can avoid the Dom0 system cache.

For fully virtualized VMs without the Novell driver pack this unfortunately does not work, here the cache must mostly be accepted. However, mostly does not mean completely, at least Windows VMs can influence the »spontaneity« with which write is effected. This is possible by disabling the write cache within the VM for the virtual IDE HDs. Consequently, write operations are performed by the Dom0 in a decidedly more spontaneous way (as can be seen with »iostat« based on the increased write accesses). Therefore, risks with regard to data integrity can be reduced, but not eliminated.

This shows how thoroughly important the Novell driver pack is for fully virtualized VMs. Only with this package is it possible to force a fully virtualized VM to bypass the cache, therefore it should not be regarded as an option, but - inasmuch as it is available for a certain operating system - more as mandatory.

When comparing the forms of virtualization the following cache variants are considered for the fully virtualized VMs:

Full (min. SC / DC on)	System cache minimized, Write-back cache of the emulated IDE disk enabled (standard setting for Windows)
Full (min. SC / DC off)	System cache minimized, Write-back cache of the emulated IDE disk disabled
Full (max. SC / DC off)	Write-back cache of the emulated IDE disk disabled.

Consideration of the setting »(min. SC / DC off)« is given preference below. This setting provides the most realistic performance values, because it reflects the situation of a host with a high load with regard to the size of the system cache and also minimizes the danger of data loss during write as a result of a crash.

Measurement environment

All the tests concerning disk I/O were performed in a VM and in a native system of the following configuration:

Number of CPUs	1 core
Available RAM	1536 MB
Operating system	Microsoft Windows 2003 R2 Enterprise x64 Edition (SP2)

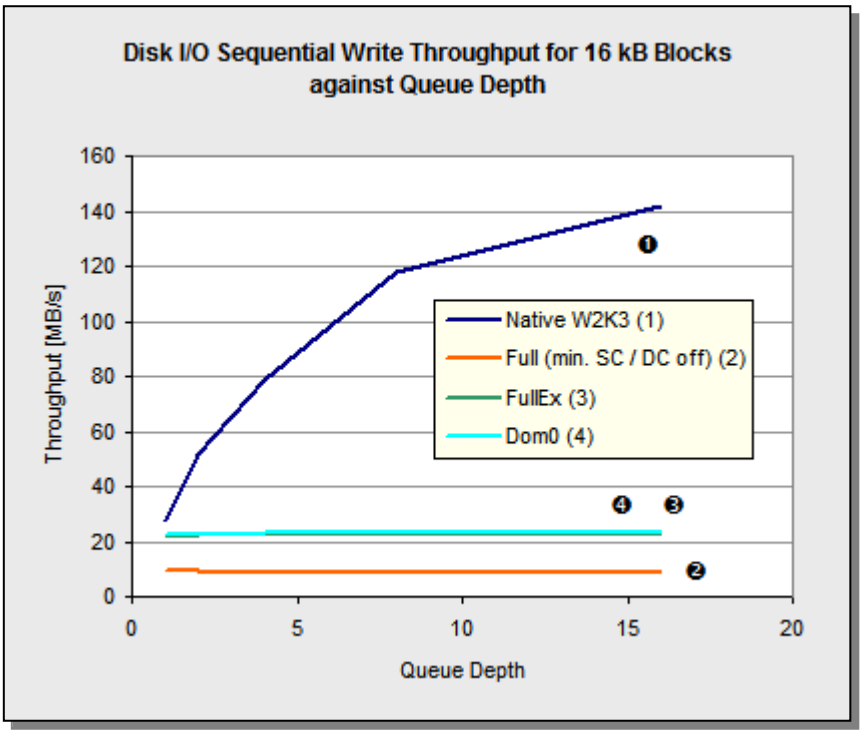
Windows was selected as the operating system, because both the operating system and the measuring tool Iometer under Windows are known to support asynchronous disk I/O. Problems of the virtualization layer or the host with asynchronism would be provable in this way.

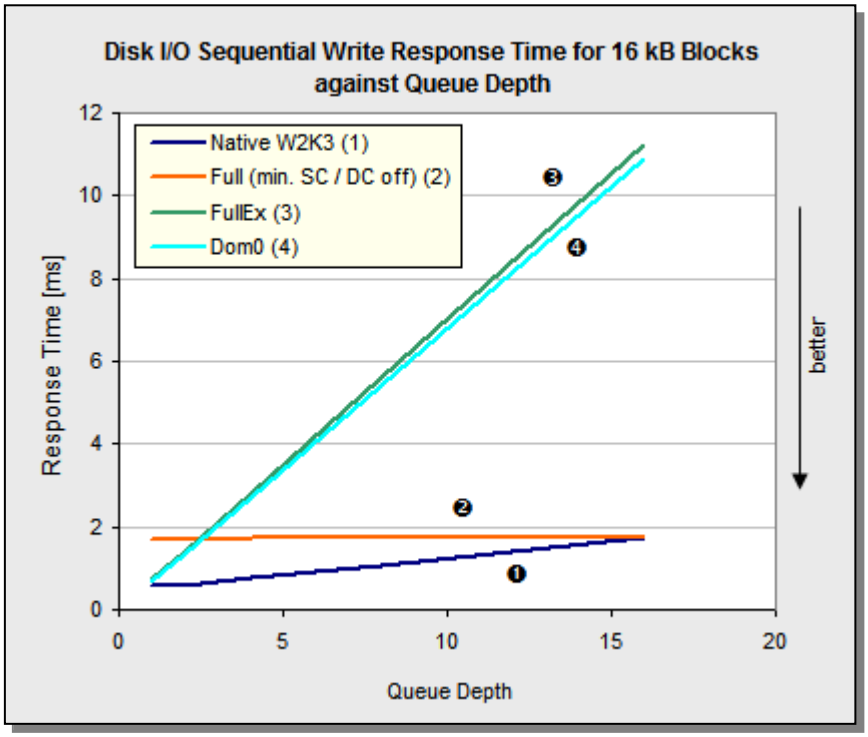
To determine the data throughput we will consider the following access patterns (in Iometer: »Access specification«):

Access pattern	Block sizes	Read/Write share	Random share	Queue depths
Sequential Read/Write	512 B (minimum) to 64 kB (maximum) in steps to the power of 2	100% write with all block sizes; then 100% read with all block sizes	0%	1,2,4,8,16
Database	8 kB	67% Read, 33% Write	100%	1,2,4,8,16
File server	64 kB	67% Read, 33% Write	100%	1,2,4,8,16

The detailed dependencies of the virtualization overhead with the disk-I/O data throughput on the form of virtualization, on the access profiles and on the system cache variants are relatively complex so that they need to be examined and discussed step-by-step in the following sections.

Asynchronism





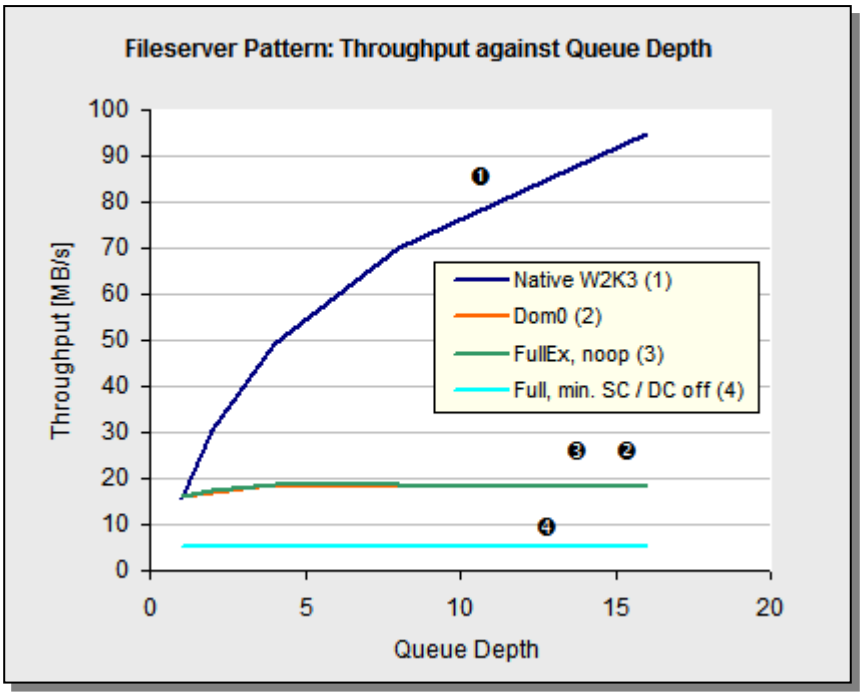
This series of measurements examines the influence of queue depth, that in other words is the parallelism of I/O accesses.

The diagrams opposite show examples of data throughput for sequential write with 16 kB block size.

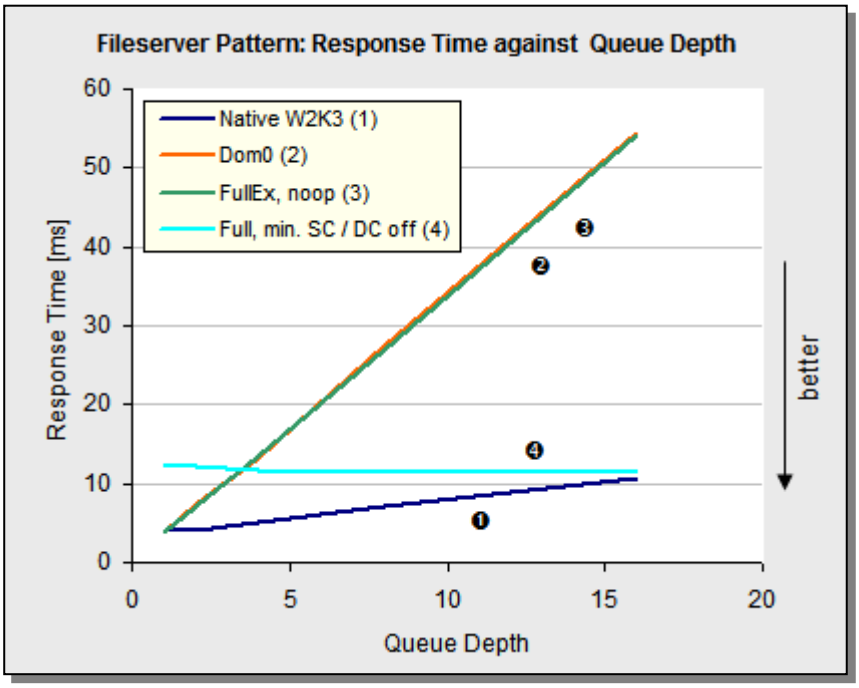
If you look at the first diagram, which shows the data throughput with an increasing queue depth, you can see that the data throughput for the VMs of the two measured forms of virtualization does not increase with queue depth, but is stagnating. The same applies for the Dom0 (and would also be seen for a native SLES10). For queue depth 16 the native W2K3 throughput with 141 MB/s is almost maximum throughput, whereas the 'FullEx' VM with 22 MB/s achieves just under 15% of the native throughput. This is relatively dramatic, since modern storage systems depend on an asynchronous supply of I/O requests in order to compensate their higher latency and utilize the bandwidth of the disk connection.

In the second diagram, which represents the response time, you see that the latter increases almost on a linear basis with queue depth for the 'FullEx' VM. Iometer calculates response time from the point in time immediately before generation of the I/O operations. Waiting times within Iometer's own loop logic are not included. Thus the response times that increase on a linear basis with queue depth prove that the asynchronous requests are transferred completely asynchronously from the I/O layer both within the VMs and in the Dom0, without blocking the caller Iometer. Therefore, serialization of the I/O orders must take place in a later step, as a result of which a request tailback occurs.

Very similar behavior results if - instead of sequential accesses - typical access patterns of a file server and a database server with block sizes of 64 kB and 8 kB respectively and in each case random accesses with $^1/_3$ write and $^2/_3$ read share are considered. »noop« is recommended as the I/O scheduler for all VMs (see section »IO Scheduler«).



The throughput of a native system with Windows Server 2003 starts at a queue depth of 1 with 15.8 MB/s and increases by queue depth 16 to 94.6 MB/s; in other words it makes clear use of the scope of the asynchronous I/O. Dom0 and a 'FullEx' VM also start at queue depth 1 with about 16 MB/s, but show no increase in throughput as queue depth rises. The 'Full' VM starts with 5 MB/s and shows almost no increase in throughput with rising queue depth. The effects here are in principle the same as with sequential accesses with the data throughput generally being somewhat lower for random accesses than for sequential accesses.



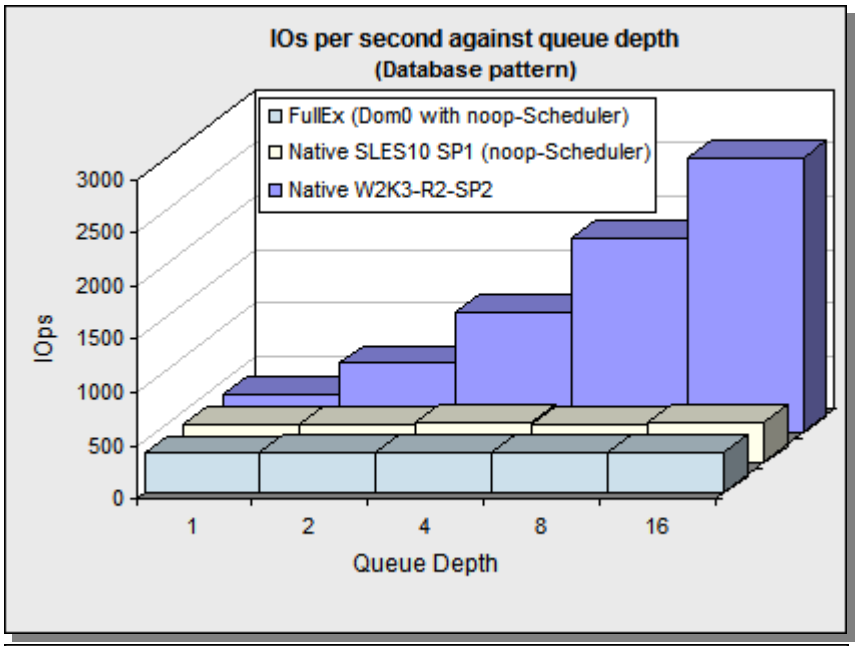
The response times for the four situations being considered are for queue depth 1 as calculated from the data throughputs. In other words, for the native measurement, for Dom0 and for the 'FullEx' VM they amount to 3.9 ms, and the highest value with 12.4 ms is for the 'Full' VM.

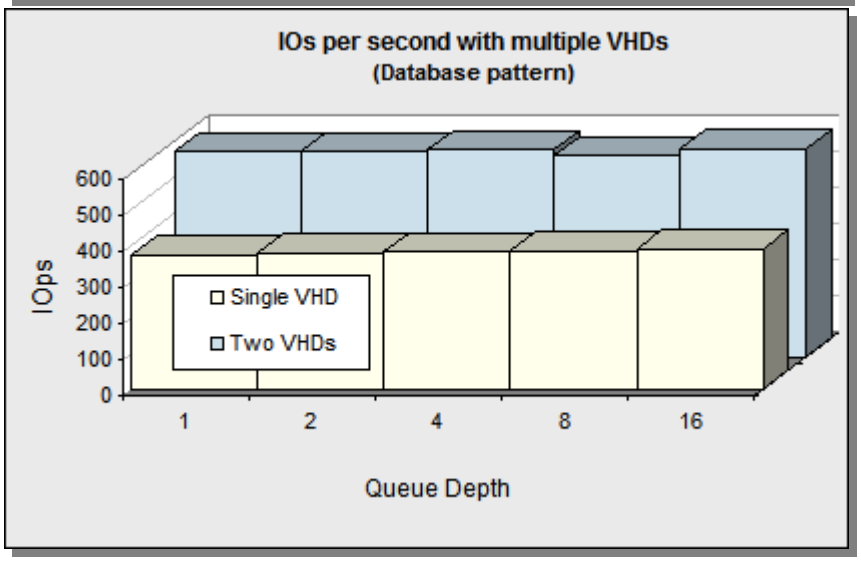
In all cases - except for the 'Full' VM - the response time then increases as calculated from queue depth and throughput, in other words it is the greatest for the 'FullEx' VM at queue depth 16.

However, with the 'Full' VM the response time for a higher queue depth is in contrast to the throughput implausible, because it remains approximately constant despite rising queue depth (that is to say more requests for the same disk).

As regards quality all the statements are just as applicable for the access pattern of a database with a block size of 8 kB in comparison with 64 kB with the file server - this is why they are not depicted in a separate diagram. Quantitative differences are on account of the smaller block size not quite so pronounced.

This shows that although a VM is always in a position to control a VHD asynchronously (proven by the linear increase in mean response time with growing queue depth), a kind of »serialization« takes place at a later level, through which the originally asynchronous I/O operations are only transferred to the storage system synchronously.



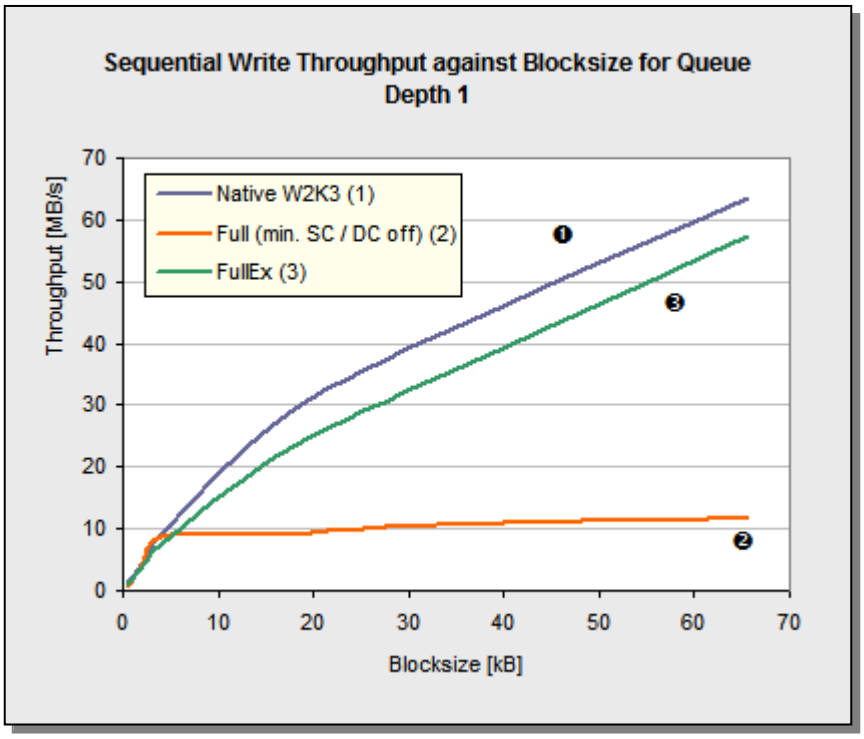


Not only are the VMs affected by this »serialization«, but also programs that run within the Dom0 and on a native SLES10 and asynchronously generate their I/O operations from an individual thread. However, as section »IO Scheduler« shows they have the option of achieving a certain parallelism of I/O operations by generating the I/O operations with several different threads instead of from one thread only. The VHDs of the VMs always have such a thread; for each VHD a dedicated kernel thread is generated in the Dom0, which performs the I/O operations in an acting capacity for the VMs. It must therefore be assumed that »serialization« takes place on the level of a Dom0 thread. This assumption is in principle confirmed by the fact that a VM can achieve an increase in throughput in a similar way to the Dom0, by working in parallel on different VHDs and consequently using several »VHD threads« indirectly. Here the VHDs may also be on the same physical disk. At the same time this excludes the physical disk as a serialization criterion.

The result of this is the recommendation for applications sensitive to disk I/O to distribute the disk load over several VHDs if possible. In order to simplify the application configuration these VHDs could for example be consolidated to form a single logical disk by means of a SW-RAID as a striping set (RAID0).

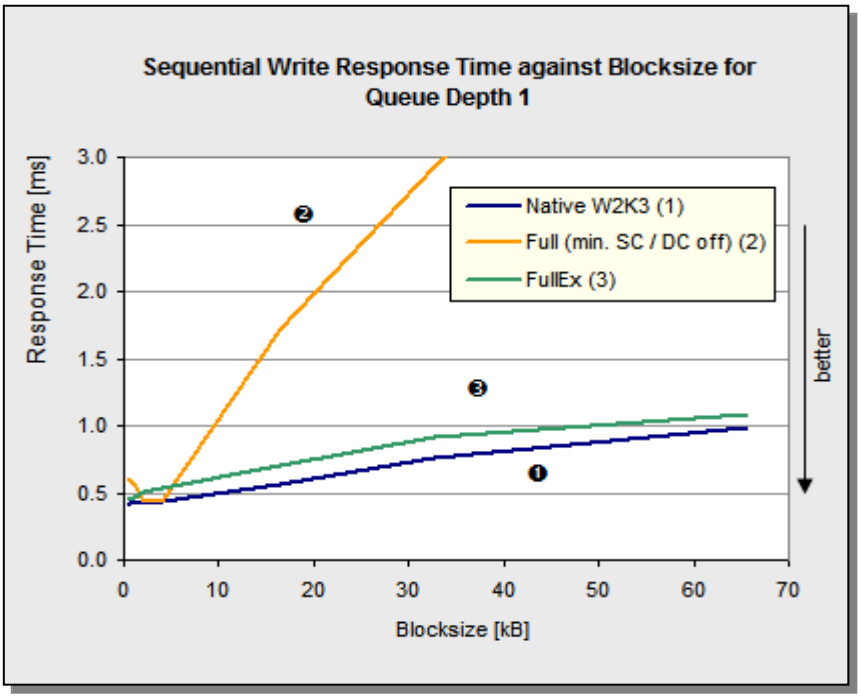
Data throughputs

We will now consider the influence of block sizes for disk I/O. As we have already learnt before that XEN serializes all the I/O requests, it is sufficient to consider a queue depth of 1, although with a native system the maximum throughput only becomes possible with higher queue depths.



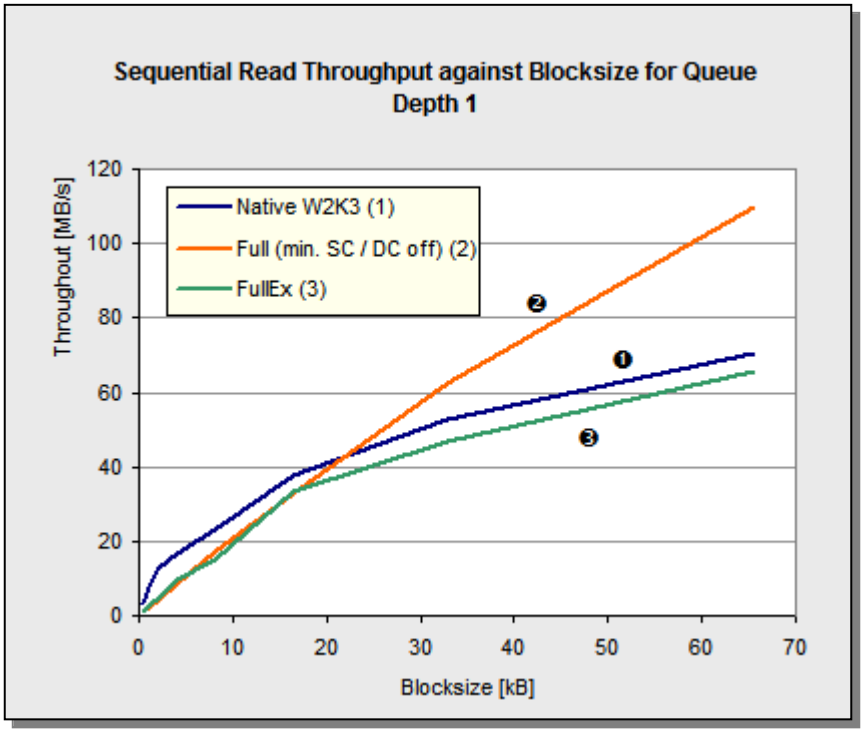
In the first diagram, which shows throughput against block size, it can be clearly seen that the native system has the best throughput for all block sizes. The throughput of a 'FullEx' VM increases somewhat more slowly and with block size 16 kB and higher runs at a constant distance of about 6 MB/s below the native system.
Up to block size of 4 kB throughput increases for the fully virtualized VM to the same degree as the native W2K3. With higher block sizes there is almost no further increase in throughput.

And with the response time the native system also has the best values, namely the lowest ones. For the block size of 512 Bytes a pure response time of 0.41 ms is measured for the native system (that is the typical write latency of the storage system used) compared with about 0.46 ms for a 'FullEx' VM. Here you can clearly see the additional time required to pass through the virtualization layer, because the influence of block size can be ignored at 512 Byte. As the block size rises, you can see a slight increase in the response times for 'FullEx' and 'Native'. The distance between the two response time curves grows to about 0.1 ms.



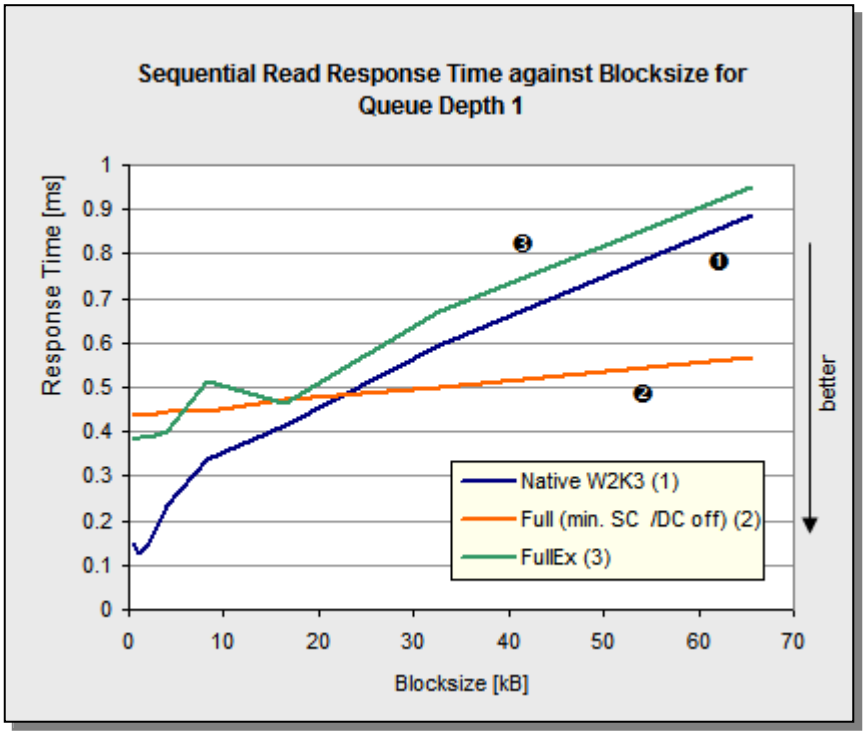
With 'Full' (»min SC / DC on«) additional read operations are observed for small block sizes up to 2 kB and as a result an additional 0.16 ms in the response time. Above 4 kB block size the response time increases rapidly, which fits precisely with stagnating throughput. Both peculiarities of the 'Full' VMs are caused by the special way the IDE disk emulation works. With an disabled disk cache in the VM this emulation always performs the write operations to the physical disk in 4 kB blocks. For orders from the VM with smaller block sizes this means that they have to be complemented to 4 kB through read from the disk. With block sizes more than 4 kB the blocks are broken down into portions of 4 kB each and these are then written in serialized form. This explains the drastically increasing response times.

And now the behavior with sequential read is to be considered. Here we will also be satisfied with a queue depth of 1, although a native system that supports parallelism would achieve a considerably higher throughput.



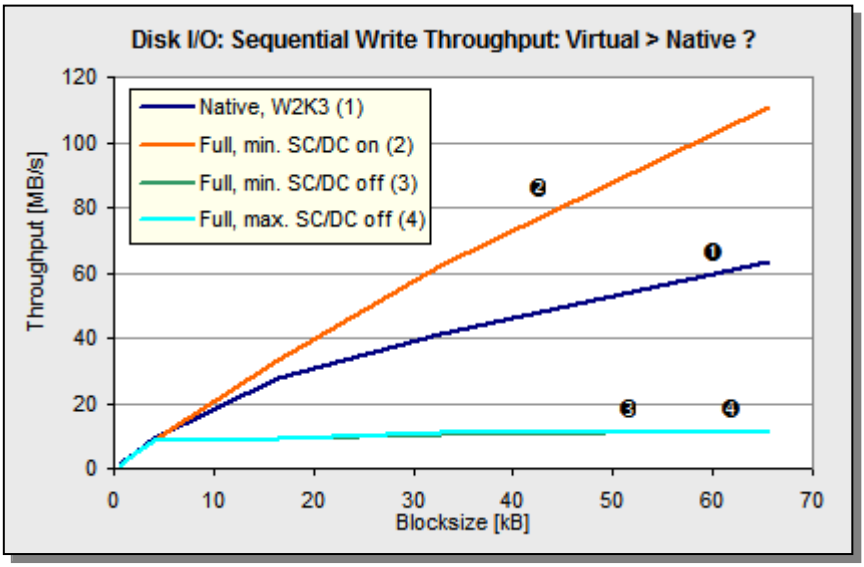
With smaller block sizes of less than 8 kB 'FullEx' and 'Full' are firstly at a disadvantage in relation to the native W2K3. For example, with 4 kB the throughput for the two VM types is only half as large as with the native system. Then read throughput increases with block size, but levels out with higher block sizes approximately like the shape of the square root function. With block size 16 kB and higher the curve of the read throughput for 'FullEx' runs at a constant distance of about 4 MB/s below the data throughput of the native system.

With the 'Full' VM and block size of about 20 kB and higher we can see for all three cache variants a stronger, almost linear growth in read throughput that is quantitively dependent on block size up to about 110 MB/s compared with a maximum of 70 MB/s for the native system. On the storage system it could be verified that the 'Full' VM reads with a queue depth > 1, although the original I/O request in the VM only has queue depth 1. With read I/O this is a sensible optimization action of the XEN disk I/Os. Read-Ahead caching cannot be excluded, either. Consequently, a higher read throughput is achieved for queue depth 1 (but not for the relevant larger queue depths) than with a native system, which does not autonomously implement such optimizations.



For the response times the 'FullEx' VM and 'Full' VM do not come below a response time of 0.38 ms for small block sizes, whereas with the native system this is at best only about 0.13 ms. With about 16 kB and higher the response time curve for the 'FullEx' VM runs at a constant distance of about 0.05 ms above the curve of the native system. For the 'Full' VM the response time curve is very flat, the times only deteriorate from 0.44 ms to 0.57 ms. And mathematically, this is clear due to the almost linear course of the throughput curve.

Sequential write with full virtualization faster than with native?



It can already be seen above in the throughput curve of the 'Full' VM for sequential read that the 'Full' VM has a higher throughput than the native system for large block sizes. With sequential write from a 'Full' VM, cache variant »min. SC / DC on«, queue depth 1, a similar effect occurs: for block sizes above 16 kB the »virtual« throughput overtakes the »native« throughput and is at 64 kB apparently 80% higher.

In contrast to read accesses, in which caching can have a positive impact on throughput, caching with write accesses, as is the case here due to the system cache of the Dom0, is not always desired because it can endanger data integrity. This should be taken into account in performance comparisons between forms of virtualization.

Maximum data throughputs with queue depth 1

The following table shows the maximum data throughputs in MB/s achieved with queue depth 1. They were all achieved with sequential disk I/O via Iometer with a 64 kB block size. The maximum achievable throughput (as a result of a higher queue depth) for the storage system used is 152.7 MB/s (write) and 173.1 MB/s (read). The value marked with an asterisk (*) in the table is not real, because in this case the data is merely in the system cache of the Dom0 (see above).

Disk I/O direction	Native W2K3	'FullEx' VM	'Full' VM		
			min. SC / DC off	min. SC / DC on	max. SC / DC off
Write	63.6	57.4	11.8	(*) 110.8	11.8
Read	70.4	65.9	109.9	109.8	118.4

Database server application scenario

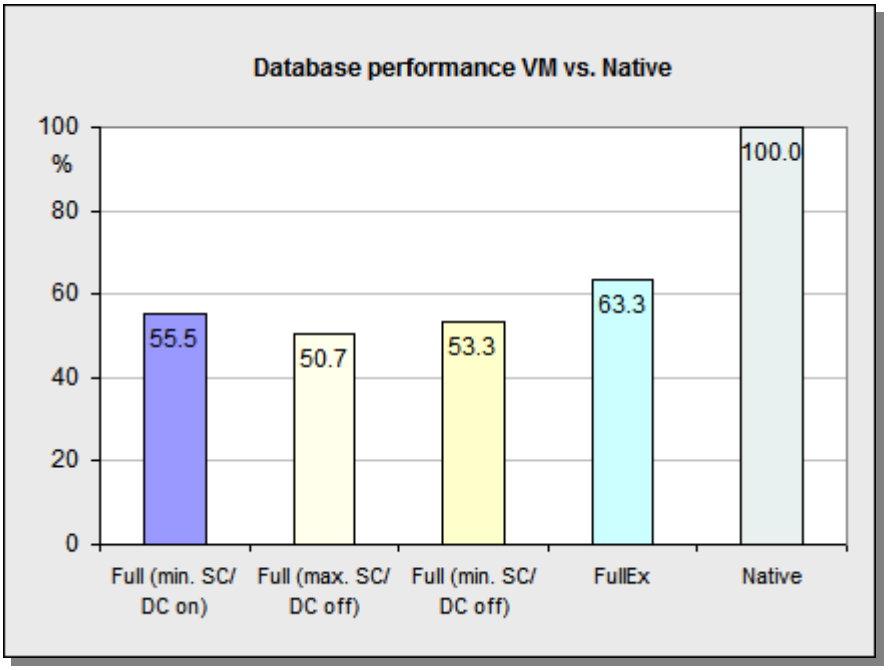
On account of the missing parallelism the VMs only achieve a similar throughput to the native system with queue depth 1, in other words without parallelism. In reality, however, this must not have such an extreme impact, because for example with a server application the I/O operations are frequently the consequence of a superior, possibly CPU-intensive transaction. Merely as a result of this a considerably more favorable load profile can arise for the disk-I/O subsystem. However, a cache within the VM that is possibly managed by the system or by an application can also have a positive influence on load behavior.

Therefore, we will consider a typical database application, as represented by the benchmark SysBench. For this purpose, we will compare the VMs and the native system with the following configuration

Number of CPUs	1 core
Available RAM	1536 MB
Operating system	Microsoft Windows 2003 R2 Enterprise x64 Edition (SP2)
Database	Microsoft SQL Server 2005
Benchmark	SysBench 0.3.3

With the native system the configuration was established via appropriate parameters in the configuration of the boot loader (boot.ini) of Windows.

The diagram opposite shows that in a complex application environment the missing parallelism has a considerably less serious impact than is the case with pure I/O load measurements. One reason for this is that very effective caching takes place within the VMs in the application scenario. This only becomes evident through analysis of the I/O requests. SysBench generates disk I/O with a read share of 60%, but according to »xentop« only 3% of the disk-I/O operations are read operations in the Dom0, the missing operations are consequently satisfied from the cache.



In this way, the VMs profit from the fact that the missing parallelism is only enforced by the serialization that takes place later in the Dom0 and that they are thus still in a position to initiate the I/O operations asynchronously. With a converse read:write ratio this would have had a stronger negative impact, the VMs would then have been obliged to wait for the read data. Since according to both »xentop« and the Windows Performance Monitor the utilization of the VMs is also 100% in all variants and in addition the Windows Performance Monitor only shows a mean disk queue depth of 0.3, it can be assumed that with this load profile the missing parallelism is not the reason for the significant losses in performance compared with the native system.

The behavior of the 'Full' VMs is remarkable. They generally show lower performance levels, but the fact that precisely the configuration with a large Dom0 cache offers the lowest performance is an indication that caching in the Dom0 is with regard to performance then at least counterproductive when effective caching also takes place within the VM.

Network

After disk I/O we will now deal with the other important I/O component - the network. Maximum possible data throughput is also to be considered first here and in a further analysis the consequences are to be examined in a real application scenario.

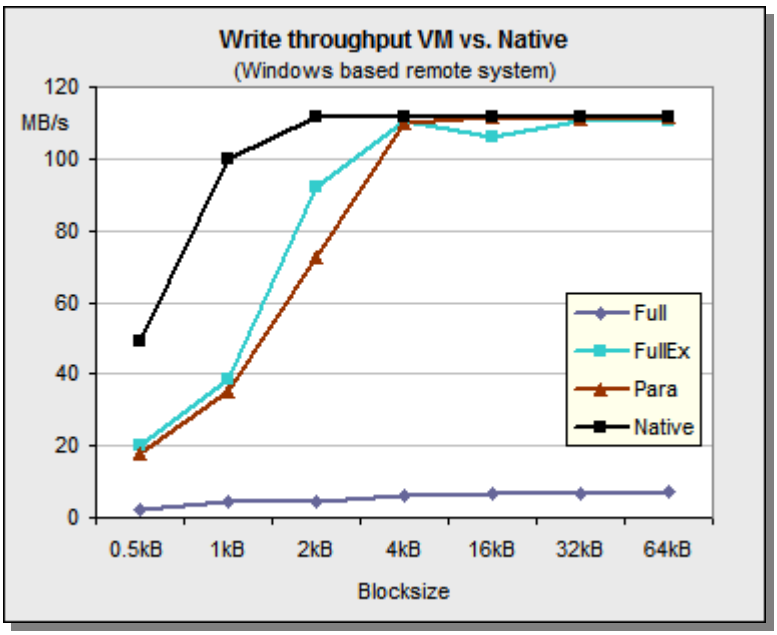
The three forms of virtualization 'Full', 'FullEx' and 'Para' as well as a native system are to be considered here. The VMs were identically configured as follows:

Operating system	SLES10 SP1 64-bit
Number of CPUs	1 core
Available RAM	1536 MB

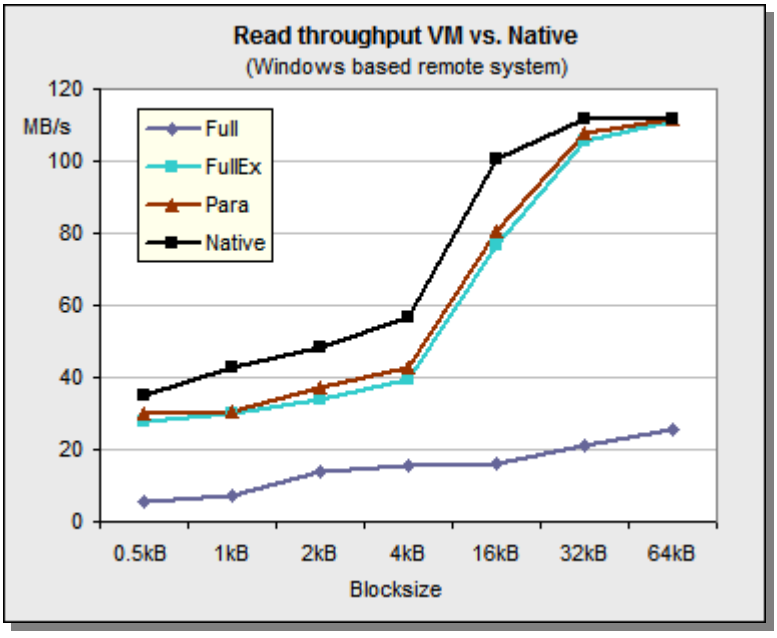
An identical configuration was established for the native system via appropriate parameters in the configuration of the grub-boot loader.

Data throughput

Iometer was used to determine data throughput. Here an Iometer instance on the system to be measured (SUT) exchanges data with different block sizes with a second Iometer instance on a separate system.

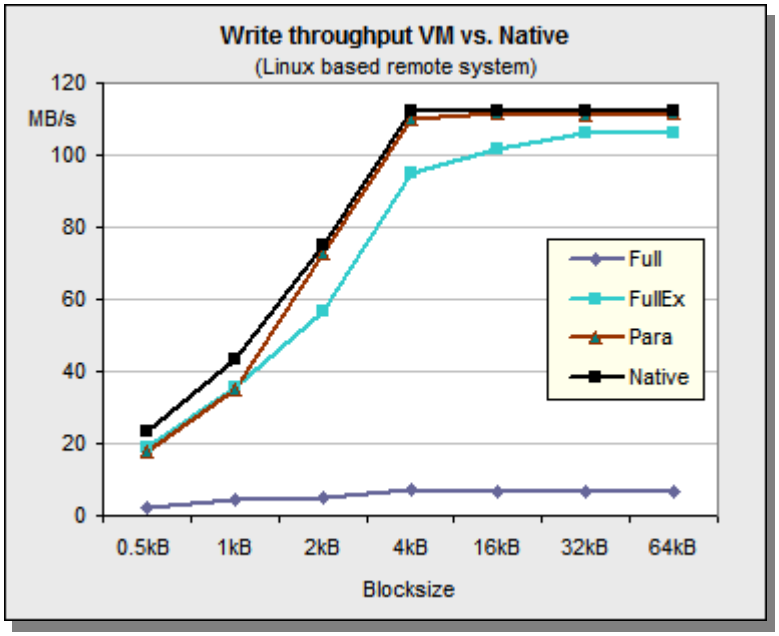


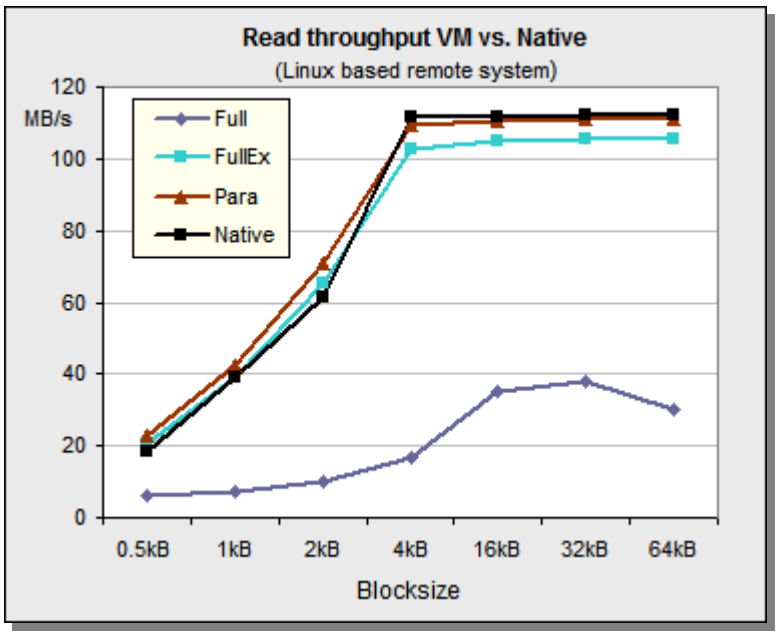
As with disk, network throughput also very much depends on the type of operation, that is to say whether data is sent or received. If you consider the write case (SUT sends), the native system achieved the maximum possible data throughput as early as with a block size of 2 kB. With 1 kB this would not have been possible due to the physics of the underlying network. Both the 'FullEx' and the 'Para' VM are also in a position to achieve the maximum throughput, even though slightly delayed for block size 4 kB and higher. The behavior of the 'FullEx' VM is remarkable; despite full virtualization it shows a better performance in the range below 4 kB, and a considerably better performance than with the para-virtualized VM for 2 kB. In return, throughput declines by up to 10% with 16 kB. With only 7 MB/s the performance of the 'Full' VM remains way below the maximum possible throughput of 111 MB/s.



If you consider the read case (SUT receives), the throughput of the native system is also clearly reduced compared with the write case. 'Full' VM is the exception - here it clearly shows better throughputs than for write. However, it still does not achieve the lowest throughput value of the other systems. Here the 'FullEx' VM now shows a performance behavior very similar to that of the 'Para' VM. Both the 'FullEx' and the 'Para' VM can with a block size of 64 kB provide the same throughput as the native system; however, considerable differences can be seen in the range below 32 kB.

With network throughput also depends on the implementation of the TCP/IP stacks of the communication partners. The above data throughputs were determined against a system with Microsoft Windows 2003 R2 Enterprise x64 Edition (SP2). A somewhat different behavior results with a counterpart based on a 32-bit Linux SLES10 SP1. Although in this case only lower throughput was achieved in the write case (SUT sends) with block sizes below 4 kB than with the Windows counterpart, the throughput in the read case (SUT receives) nevertheless rose clearly in this range Furthermore, both the 'FullEx' and the 'Para' VM showed better throughput than the native system in this range. Unlike with a Windows counterpart, the 'Para' VM with a Linux counterpart generally achieved better throughput than the 'FullEx' VM.

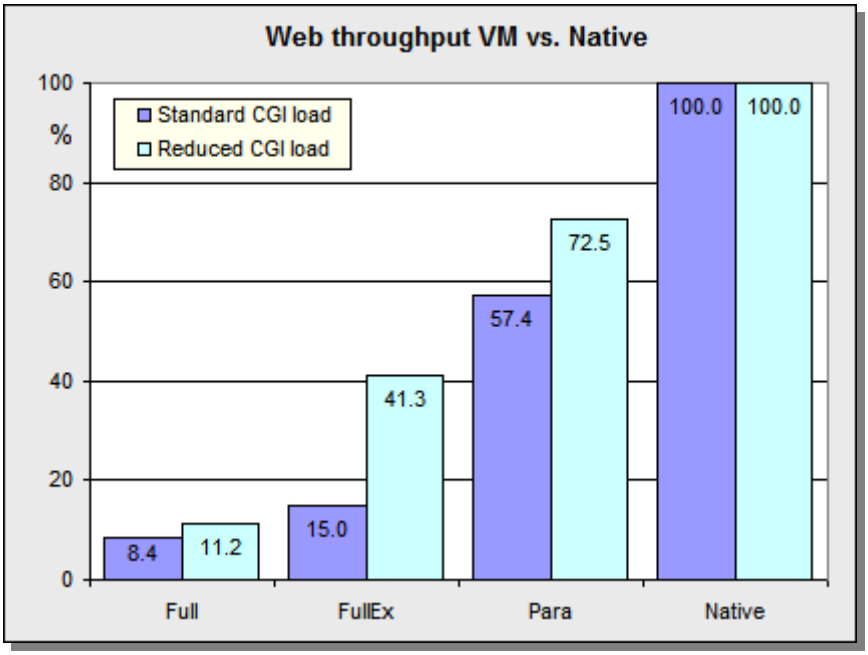




Web server application scenario

The analyses of the network data throughput certify a performance for both the 'FullEx' and the 'Para' VM that is of a virtually native level for write operations. However, load generation was effected without the time and effort involved in a server application on account of superior transactions. In order to assess the performance behavior of an application a typical web-server environment is considered on the basis of Apache 2 and of the »WebBench 5.0« benchmark [L15].

The following diagram shows the relative web throughput of the VMs related to the native system. It shows that very considerable losses in performance result in the web-server environment independent of the form of virtualization. On account of the data throughput measurements this was to be expected for the 'Full' VM, but not for the 'FullEx' and the 'Para' VM. The cause for the clear losses lies in the definition of the WebBench load profile. This defines that 16% of all HTTP requests and 2% of all HTTP-SSL requests on the web server start a CGI program. However, the start of a process within a VM is on account of the virtualization of the MMU (Memory Management Unit) considerably more complex than on a native system. To illustrate the impacts the measurements were therefore performed with the standard profile and a modified profile without the 16% HTTP-CGI requests. With the standard profile the 'FullEx' VM only achieved a relative throughput of 15%, the 'Para' VM nevertheless achieved 57.4%. With the modified profile the 'FullEx' VM was now on the contrary able to increase its relative throughput very clearly to 41.3%, but is still a long way off from the 'Para' VM, which was able to improve its relative throughput to 72.5%. The clear gap that still exists to the native system can be explained by the transfer sizes of the HTTP requests. According to the modified profile 31% of all accesses required files with a size of less than one kilobyte, for a further 17% of accesses the file size was below 2 kB. However, with the data throughput network measurements both the 'FullEx' and the 'Para' VM already showed a 2.7-fold smaller throughput with such transfer sizes than the native system.



Scaling

After intensively discussing the performance of a single virtual machine in the previous sections, the overall performance of several VMs running simultaneously on the same physical system is now to be considered. Of special interest here is the development of performance when the load caused by the VMs is increased step-by-step. This functional relation is designated in the context of virtualization as »scaling«.

In order to determine scaling the benchmark kit vServCon [L10], [L11] was used, in which one or more sets (also known as »tiles«) of application benchmarks are run in parallel and their overall performance (expressed as a »score«) is determined.

A tile is made up of three VMs, which reflect the following application scenarios:

Java server	simulated by the benchmark SPECjbb2005
Database server	simulated by the benchmark SysBench
Web server	simulated by the benchmark WebBench

Identically configured, these have in each case already been used for the measurements of the application scenarios. A virtual CPU was allocated to each VM; 2048 MB of memory was allocated to the Java VM and 1536 MB respectively to the other two VMs within a tile.

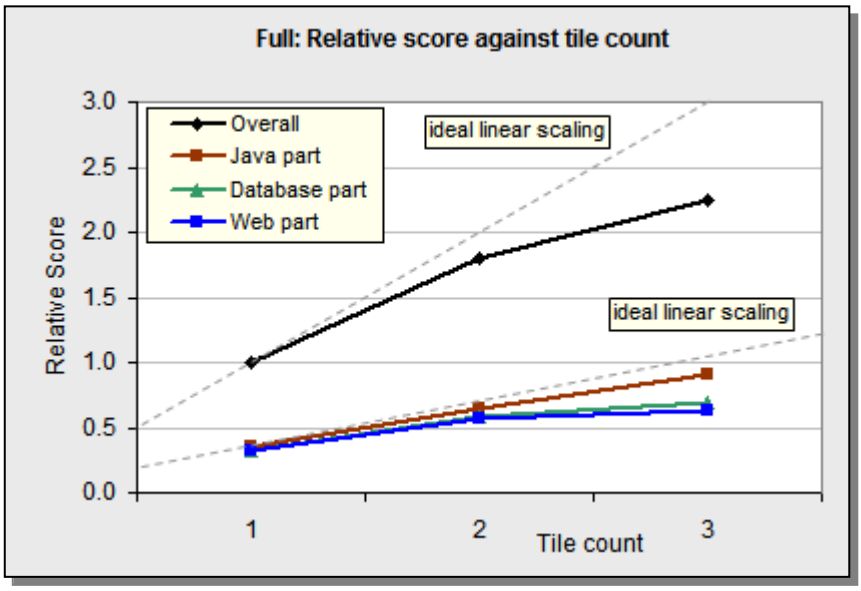
For each form of virtualization performance was determined with a load with one, two and three tiles.

On account of the test platform with eight physical cores overall good scaling is to be expected in the measurements with two tiles, because eight native CPU cores are available for the total of six necessary virtual CPUs. On the other hand, a genuine overload situation is to be expected with three tiles, because on the one hand a native CPU core is already missing due to the CPU configuration of the VMs and, in addition, a considerable CPU load in the Dom0 has to be overcome on account of the I/O activities of the VMs. The following diagrams with the measurement results confirm this presumption precisely. As long as sufficient system resources are available, good performance scaling can be observed. Depending on the form of virtualization scaling is between 1.8 and 1.9 for two tiles. However, if a configuration exists, in which more virtual resources are planned than are physically available, competitive displacement arises and scaling in the case of three tiles is only between 2.2 and 2.5.

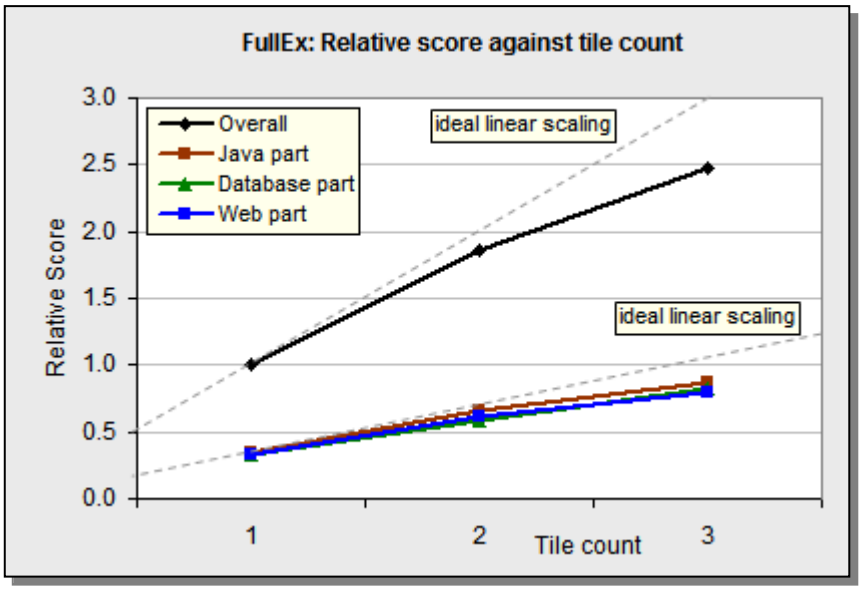
The following diagrams show the scaling for different forms of virtualization and application scenarios in detail. In addition to the scores for the complete tiles, the diagrams also show the pro-rata scores for the three types of VMs (Java server, database server and web server). As a means of orientation for scaling quality dashed lines, which would represent ideal linear scaling of the overall and pro-rata scores, are drawn through to zero.

Independent of the form of virtualization, the Java server shows the best ability for scaling, followed by the database server. The web server reacts most sensitively to resource bottlenecks. The worse scaling behavior of the I/O-dependent VMs, database server and web server, compared with the JAVA servers for the three tiles has the following reason: in addition to a shortage in their direct CPU resource, the former suffer from the virtualization-specific and general consequential effects of increased I/O orders.

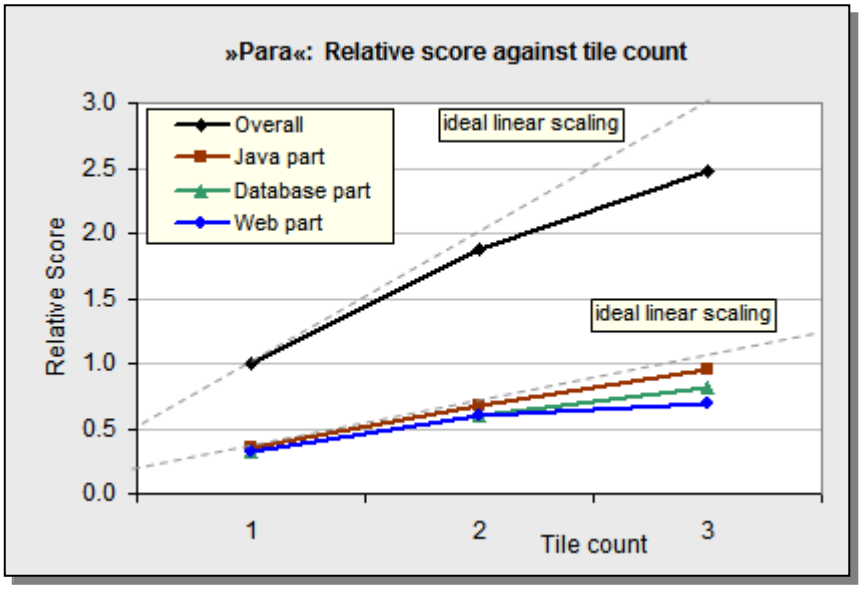
As expected, for fully virtualized VMs the scaling behavior of the I/O-dependent VMs with three tiles is the worst compared with the JAVA servers. The relative score of the database server and web server VMs hardly increases any further from two to three tiles. The reason is the in comparison highest overhead of this form of virtualization.



With the 'FullEx' VMs the scaling behavior improves considerably on account of the Novell driver pack and effectively reaches the level of the para-virtualized VMs. With three tiles saturation level begins to appear. All three server types show approximately the same scaling behavior.



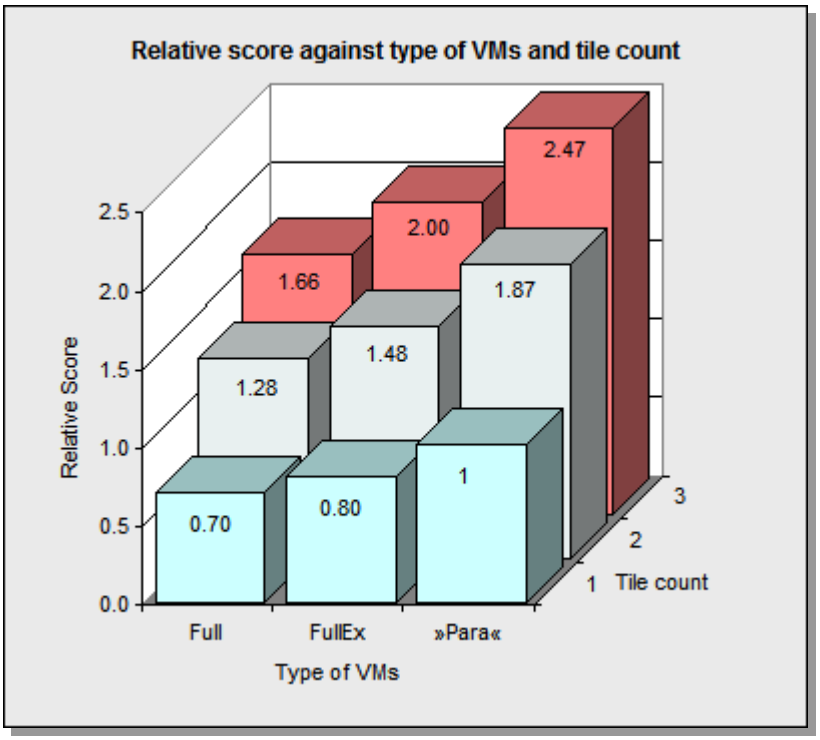
A series of measurements with para-virtualized VMs only is not possible on account of the database VM based on Windows. To at least achieve complete para-virtualization for I/O the fully virtualized database VMs were used together with the Novell driver pack. Despite this restriction a good scaling relationship of 1.87 is still achieved with two tiles. The leveling-off of the curve with three tiles to 2.47 was to be expected on account of the shortage of CPUs. Of particular note in this measurement is the greater decline in performance in the para-virtualized web VMs compared with the only fully virtualized database VMs.



Overall performance of VMs dependent on the form of virtualization and the number of tiles

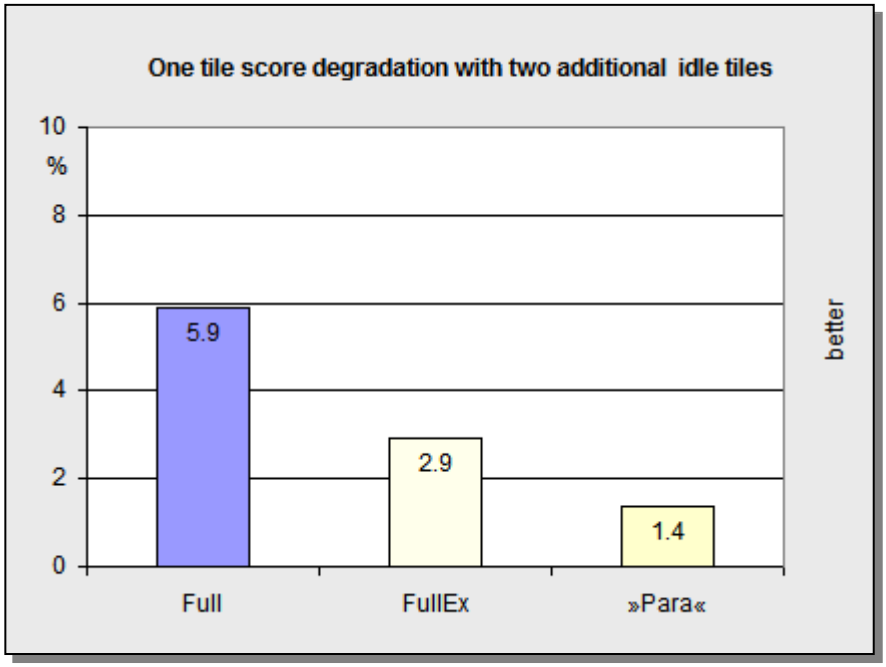
The diagram opposite compares both the scaling and the performance of various forms of virtualization. For the purpose of comparability relative scores, which are related to the score of an individual »Para« tile, are provided.

This comparison is in this respect important, because in server consolidation several virtual servers are migrated to one physical server. You can see from the diagram that the score difference between one 'Full' tile and one »Para« tile is 43%. In the event of three 'Full' tiles compared with three »Para« tiles the difference grows to 49%. Compared with the 43% for one tile, the 49% for three tiles includes both influences, namely the difference in the virtualization forms in themselves and the different scaling.



Influence of idle VMs

Idle VMs - in other words VMs that run without a real »workload« - also represent a load for the virtualization layer, because they nevertheless take part in normal scheduling and accordingly have to receive CPU time assigned by the scheduler of the virtualization layer on a regular basis. As a consequence, the VMs which are subject to load show reduced throughput. The diagram opposite shows how the score of a single active tile changes if six idle VMs also run completely unloaded. In comparison with the measurements without the »idle« VMs, losses in performance can be seen in all forms of virtualization. With the fully virtualized VMs ('Full') the score of the active tile is clearly reduced by 5.9% (0.98%/VM) compared with the tile that was measured without the idle VMs. If the Novell driver pack is used ('FullEx'), the score reduction is halved (0.48%/VM). This is remarkable inasmuch as the additional VMs performed no apparent I/O activities at all and thus it must be assumed that the unavoidable I/O background activities of the operating systems within the VMs are already responsible for the considerably higher losses of the fully virtualized VMs. The measurement with the para-virtualized tiles has the lowest score reduction with only 1.4%. With this measurement it was not possible on account of the Windows-based database VMs to measure only para-virtualized VMs. With due regard to the two 'FullEx' database VMs the share in this case of the four para-virtualized VMs of the score reduction is only 0.43 (0.11%/VM).



Summary:

The performance of a VM under XEN depends a lot more on the type of application scenario than with a native system, especially with regard to I/O. On account of the distinct differences in I/O performance between a native system and a VM the field of operation for XEN lies in the server consolidation of older existing systems. For the virtualization of current high-load servers, especially those with a high disk load, the performance analyses have shown that XEN is less well suited in this regard.

Both from a performance view and with regard to data integrity it is advisable to always configure VMs in such a way that their virtual hard disks bypass the system cache. When using full virtualization this is only possible in connection with the add-on product »Novell Driver Pack«; use of this pack should not be ignored, because a fully virtualized VM otherwise already has considerably higher CPU requirements in an idle state.

In some cases, it is possible to observe better I/O measurement values in a VM than with the native system with synthetic benchmarks. The reasons for this are on the one hand cache effects that arise in the overall not-to-be-recommended situation of a fully virtualized VM without a driver pack, and on the other hand the support of the VMs through the decoupled functioning Dom0, which has additional CPU resources. However, this better I/O performance with VMs in certain cases should hardly affect complex applications, especially not if the resources for the Dom0 become more scarce due to the operation of several VMs.

With disk I/O the I/O architecture of the Dom0 considerably influences performance. Applications that depend on really asynchronous processing of their I/O operations are less well suited for virtualization or should be supported with a specialized configuration of the virtual hard disks (e.g. SW RAID within the VM). Furthermore, more attention should be paid to the configuration of the I/O schedulers; the »noop« scheduler is as a rule the best suited scheduler.

Whether an application with XEN can be sensibly virtualized also always depends on its run-time share in the kernel mode. Execution of code in kernel mode always causes an additional virtualization overhead compared with user mode; this particularly affects full virtualization (see section Web server application scenario).

If it is possible with the operating system in the VM, a para-virtualized VM should be selected, as this always has the lowest overhead compared with the native operating system.

Literature

[L1]	General information about products from Fujitsu Technology Solutions http://www.ts.fujitsu.com
[L2]	General information about the PRIMERGY product family http://www.primergy.com
[L3]	PRIMERGY Benchmark - Performance Reports and Sizing Guides http://ts.fujitsu.com/products/standard_servers/primergy_bov.html
[L4]	Intel Virtualization Technology http://www.intel.com/technology/platform-technology/virtualization/index.htm
[L5]	AMD-V http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8796_14287,00.html
[L6]	SUSE Linux Enterprise Server 10 Virtualization http://www.novell.com//products/server/virtualization.html
[L7]	SUSE Linux Enterprise Virtual Machine Driver Pack http://www.novell.com/products/vmdriverpack
[L8]	SUSE Patch Support Database http://support.novell.com/linux/psdb
[L9]	Iometer http://www.iometer.org
[L10]	vConsolidate http://www.intel.com/technology/itj/2006/v10i3/7-benchmarking/6-vconsolidate.htm
[L11]	vServCon - Benchmark Overview http://docs.ts.fujitsu.com/dl.aspx?id=b953d1f3-6f98-4b93-95f5-8c8ba3db4e59
[L12]	SPECjbb2005 http://www.spec.org/jbb2005
[L13]	SPECjbb2005 - Benchmark Overview http://docs.ts.fujitsu.com/dl.aspx?id=5411e8f9-8c56-4ee9-9b3b-98981ab3e820
[L14]	SysBench http://sysbench.sourceforge.net/
[L15]	WebBench http://www.lionbridge.com/lionbridge/en-us/services/outsourced-testing/benchmark-software.htm

Contact

PRIMERGY Performance and Benchmark
mailto:primergy.benchmark@ts.fujitsu.com
PRIMERGY Product Marketing
mailto:Primergy-PM@ts.fujitsu.com

Published by department:

Enterprise Products
PRIMERGY Server
PRIMERGY Performance Lab
mailto:primergy.benchmark@ts.fujitsu.com

Internet:
http://ts.fujitsu.com/primergy

Extranet:
http://partners.ts.fujitsu.com/com/products/servers/primergy