

Running Informix Dynamic Server on Linux in Xen hypervisor

First tests and results

Skill Level: Intermediate

[Martin Fuerderer \(martinfu@de.ibm.com\)](mailto:martinfu@de.ibm.com)
Software Developer
IBM

[Nicole Neuburger](#)
Student Information Systems
University of Applied Sciences Munich, Germany

17 Jul 2008

Can IBM® Informix® Dynamic Server (IDS) run on Linux® in Xen hypervisor virtual machines (VM)? Is all OS-specific functionality of IDS usable in a Xen VM? This article is a detailed summary of first experiences with running IDS for Linux in a Xen managed virtual machine. In this article, learn more about the benefits of Xen hypervisor and see how it works. Follow step-by-step guidelines to set up an actual test system, including the Xen virtualization layer. Finally, learn about the results of tests that the authors ran on their IDS environments.

Introduction

With computers using more processors and, thus, becoming ever more powerful, virtualization is a topic of growing interest. Although virtualization can be used for many different purposes, it is most commonly used to improve hardware utilization. Another use is the separation of software installations to keep them from interfering with each other and to simplify the administration.

For IBM Informix Dynamic Server, virtualization is useful to separate the data server from applications and application servers in a two- or three-tier environment. Also,

virtualization can aid in simulating environments with multiple data server instances on multiple machines, like replication scenarios with ER, HDR or MACH11.

What is Xen?

Xen hypervisor is an open source Virtual Machine (VM) manager, licensed under the GNU Public License GPL2. Originally it was started at the University of Cambridge, and then further developed by XenSource, Inc. In August 2007, XenSource, Inc was taken over by Citrix Systems, Inc. Xen hypervisor is available on several hardware architectures including Intel x86, Intel x86_64, IA64 and Power PC. Supported operating system platforms are Linux, Solaris, Microsoft Windows® and several BSD derivatives of UNIX®. Xen hypervisor is a thin software layer that is inserted between the hardware and the operating system. This setup enables the running of several different operating system instances on the same hardware.

Uses of Xen virtualization

- Improved utilization

Without the need to care for interoperability, Xen virtualization allows the deployment of completely different and independent software applications on the same hardware. If existing hardware is underutilized, the available resources can be used by virtual machines. Only when utilization reaches the maximum, does purchasing new hardware needs to be considered.

- Consolidation

In a distributed and heterogeneous environment, systems and applications can spread out of control, especially in times of fast growth. At some point, such environments become so interwoven and interdependent that they are very difficult and expensive to control and maintain. This is the time for consolidation. With Xen hypervisor, it is possible to centralize different systems (operating systems and application software) on a few machines -- making the systems easier to control and cheaper to maintain. In addition, consolidation often allows for additional savings by increasing energy efficiency (for example, on air conditioning, cooling, and electricity).

- More flexibility

With Xen hypervisor, it is easy to quickly start a new virtual machine. As long as existing hardware has the spare capacity, no purchase of new hardware, with possibly lengthy approval processes, is necessary. For example, for test purposes, a new virtual machine can be provided within

minutes, including "private" root access. Even complicated test environments like clustered machines or IDS MACH11 scenarios can be simulated with Xen hypervisor on a single machine. Service providers use virtualization to provide, on demand, complete systems to their customers or can meet changing requirements at a moment's notice.

How does Xen work?

Before Xen, virtualization typically was implemented with a microkernel. Basically, this is an operating system underneath the operating system. The microkernel contains device drivers and performs binary translation between the operating system and the device drivers. This way, the microkernel emulates for the operating system a native chipset, but this adds a significant overhead. Due to the interdependencies, a separate maintenance schedule is necessary for the microkernel and the operating system. In addition the microkernel is vulnerable to device driver failure, and the overall complexity requires a large code base for the virtualization layer.

For Xen hypervisor, the recommended virtualization method to use is *paravirtualization*. This method is state of the art, and its implementation was pioneered by the Xen development, thus giving Xen hypervisor a head start with this new technology. With paravirtualization, only the base platform, consisting of CPU, MMU, memory and low level interrupts, are virtualized. On some architectures (for example, Intel VT-x and AMD Pacifica) this capability is provided by the hardware. The device drivers are kept separate from the guest operating systems. All native Linux device drivers are supported.

As a result, the Xen code base can be small, efficient and trusted. The guest operating systems have to cooperate with Xen but can expect near-native performance. Also, with the separation of the device drivers from the guest operating systems, the release cycles are independent, and no separate maintenance is necessary.

Linux setup with Xen hypervisor

For hardware, a standard PC with one Intel Dual Core 2 Processor is used. This processor also offers the Intel VT-x functionality for hardware virtualization. Make sure you have SUSE Enterprise Linux Server (SLES) 10.1 (from Novell) as your operating system; it is installed with Xen hypervisor 3.0. Since Xen hypervisor 3.0 is part of SLES 10, only one installation is necessary. Just make sure that Xen gets installed during the installation of SLES.

To be able to test IDS's operating system specific functionality, two disk partitions are set aside for block devices to be used in the VM. One of the partitions is needed

for a file system, the other for IDS chunks on raw devices.

After installation, a VM needs to be created. Using `yast2` (yet another setup tool) as user `root`, choose **Virtualization** and **Virtual Machine Manager**. Select **New** to create a VM. In the following installation menu, select **I need to install an operating system** and then **SUSE Linux Enterprise Server 10**. Select **Paravirtualization** as the virtualization method. Since the machine has only two physical CPU cores, the VM is configured with one CPU only. This means that running VM then uses only resources corresponding to one of the two available cores, leaving the other core to be used by the base system and virtual machine manager. In theory, it is possible to configure a VM with more CPUs than there are physical CPUs (or cores) in the machine. However, this is not recommended because it causes performance degradation.

The most important commands to control a VM

- Start a VM: `xm start <VM name>`
- Access a running VM: `xm console <VM name>`
- View VM configuration: `xm list -l <VM name>`
- After configuration file change: `xm new -F <VM name>`
- Stop a VM: `xm shutdown <VM name>`

There are several commands to control the newly created VM from a terminal window. Usually these commands are executed as user `root`. The `start` command (`xm start <VM name>`) starts the VM and boots an instance of the Linux operating system in the VM. To access a system running in a VM, the `console` command (`xm console <VM name>`) is used. The current configuration of a VM can be viewed by using the `list` command (`xm list -l <VM name>`). If the configuration is changed in the configuration file, these changes are advertised to a running VM with the `new` command (`xm new -F <VM name>`). Finally, the VM can be stopped with the `shutdown` command (`xm shutdown <VM name>`). These commands can also be performed within `yast2` using the Virtualization and Virtual Machine Manager menu items.

With a VM now running on the machine, there are two instances of the installed Linux operating system. The base instance which is always running and needed for the virtual machine manager is generally named *Dom0* (domain zero). The VM is referred to by its name as more than one VM can be started. However, for this document, only one VM is started and thus simply referred to as "the VM".

For many tasks (like transferring files) it is easier to connect to the VM via a TCP/IP network connection. For this, set up your network connections by configuring your network cards for the *Dom0* and the VM. On a stand alone machine, you can use private IP addresses for this network configuration. With network connectivity set up,

you can use commands like `ssh` and `scp` to connect to the VM and transfer files between Dom0 and the VM.

Configuring block devices for the VM

To prepare for IDS chunks, some block devices are needed in the VM. To make block devices accessible in the VM, the respective device configuration needs to be added to the VM. First execute the `xm list -l <VM name> > <VM name>.sxp` command to save the current configuration of the VM in a configuration file for this VM in directory `/etc/xen/vm`. Then edit this file `/etc/xen/vm/<VM name>.sxp` to add the following entries:

- `(dev <partition>:disk)`
- `(uname phy:/dev/<partition>)`
- `(mode w)`

Example of block device entries in the VM configuration file:

```
device
(vbd
(uuid 506cebr8-0a9e-c391-b34e-58768afc2f27)
(devid 51728)
(driver paravirtualised)
(dev sda8:disk)
(uname phy:/dev/sda8)
(mode w)
(type disk)
(backend 0)
)

device
(vbd
(driver paravirtualised)
(dev sda7:disk)
(uname phy:/dev/sda7)
(mode w)
(type disk)
(backend 0)
)
```

After editing this configuration file, it is necessary to make the changes known to the VM. Do this by executing the command `xm new -F <VM name>`.

Of the two block devices configured in the above example, the first one with the partition name `sda8` is later used for IDS to configure chunks on a raw device. The second partition with the name `sda7` is used for the creation of a file system.

For the raw device, no further configuration is necessary. However, in accordance

with the standard practice for IDS, it is recommended that you create and use a symbolic link in a file system pointing to the raw device. For example, a command like `ln -s /dev/sda8 ~informix/rawdisk1` creates a symbolic link named "rawdisk1" in the home directory of user informix and this link points to the raw device /dev/sda8. When you create IDS chunks, use the symbolic link name (rather than the raw device name). Whether IDS really is using raw devices can be seen by the existence of IDS threads with the name KAIO. Check this by using a command like `onstat -g ath`.

It is recommended to do such configuration changes for the VM with the described method. It is also possible to change the configuration of a VM in an interactive way. However, changes applied with the interactive method are generally not persistent. After a shutdown and re-start of the VM, such changes are lost.

The installation of IDS in a VM is straightforward. There is no difference to installing IDS on a native SLES 10 Linux platform.

IDS's OS-specific functionality

IDS utilizes several functionalities that are specific to the operating system (OS) on which IDS is running. With Xen hypervisor being a software layer between the operating system and the hardware, it is interesting to see whether IDS can use all this OS-specific functionality while running in a Xen hypervisor VM.

- Raw devices

Different from files in a file system, raw devices are accessed directly without a software layer between the device driver and the application. Therefore, raw devices also avoid the buffering that usually is in effect for file system files. Because of that, doing a write system call to a raw device ensures that a successful return from the system call means that data really has been written onto the disk. It is not just committed to a buffering layer where it could be lost completely in case of a disruptive event (like a power cut). This is important for the transactional concept of IDS. With IDS completely in control of the disk activity, raw devices often can be used more efficiently and thus usually offer better performance.

- KAIO

I/O to disk devices normally takes some time, that is, issuing a read or write system call takes time until the I/O operation is complete and the system call returns. During this time the calling process cannot do other work as it is "stuck" in the I/O system call. Therefore, IDS always uses asynchronous I/O. Because of that, a worker process doesn't need to wait

for the I/O operation to complete --- instead, it can be doing other necessary processes at that time. Once the I/O has completed, the process will be notified. Together with the threading architecture of IDS, this enables very efficient CPU utilization.

Even more efficient than asynchronous I/O is Kernel Asynchronous I/O (KAIO). KAIO means that the kernel will do the asynchronous I/O on behalf of the calling process. When using KAIO, IDS does not need to have the extra processes doing the asynchronous I/O. This helps save overhead on those processes. KAIO is used with raw devices and allows the same level of efficiency and control over I/O operations.

- Direct I/O

A relatively new I/O method offered by the Linux operating system is direct I/O. It allows similar control over the I/O operation as utilizing raw devices, but is available for file system I/O. Because of that, raw devices are no longer a requirement for running a well-performing IDS on Linux. Also, using direct I/O instead of raw devices allows you to skip the cumbersome configuration of the raw devices.

- No aging

Normally operating systems, including Linux, decrease the scheduling priority of long running processes over their life time. This is called process aging. It favours new or short running processes with the assumption that long running processes are not expected to produce results quickly. But for a data server like IDS that is expected to be available all the time, this scheduling priority management is counterproductive. Especially when IDS is running on a dedicated machine, there is no reason why IDS's scheduling priority should get lowered as there are no other important tasks to accomplish on the machine.

To avoid this default behaviour of the operating system process scheduler, aging can be turned off for a specific process. This process' scheduling priority will not decrease over time. Instead it will always be scheduled with the same unchanged priority and thus receive enough CPU resources.

- Processor affinity

On a multi-processor (or multi-core) machine, processes need to be distributed among the available processors. With more processes than processors (the normal scenario), a process can get scheduled to run on

any processor, even when this processor is different from the one where the process was running before. When this situation occurs, a so-called "processor context switch" is necessary. It means that registers, processor memory cache, and the like need to be adjusted on the new processor before the process can run. This requires additional overhead that sometimes offsets the gain from running this process on the first available processor. Especially when IDS is running on a dedicated machine, its own processes could be scheduled in a round robin fashion on the different processors. This means the processor context switch overhead for every process of IDS is compounding.

To avoid the processor context switch overhead, it is possible on Linux to "pin" a specific process to a specific processor. By pinning a specific process, a runnable process will wait for its own processor to become available rather than migrating to a different processor that currently may be free. This action is called processor affinity. It is supported by IDS via the onconfig file parameter VPCLASS.

Testing

Test scenarios

Using the described VM configuration, tests are performed to verify that the above mentioned OS-specific functionality of IDS works correctly. For actually running a test, an IBM-internal testware named "IDStest" is used. This utility accepts specific pre-settings defined by the user to configure and start an IDS instance. It then runs several basic test scenarios including a small TPC-C benchmark scenario. However, this is not used to actually measure performance but rather as a convenient way of producing a mix of SQL statements executed by several concurrent user sessions. The scenarios are basic as they are designed to cover the rudimentary functionality and to allow automated test result analysis. Complex scenarios like user-defined types or user-defined routines are therefore not executed. Still, the testing is sufficient to judge the correctness of the OS-specific functionality within IDS.

To cover the different OS-specific functionalities, IDStest is run repeatedly with these different pre-settings:

- Normal configuration:
IDS chunks in file system (cooked files) and on raw devices using KAIO
- IDS with direct I/O:
IDS chunks in file system and on raw devices
Pre-settings for the IDS configuration:

- onconfig file parameter `DIRECT_IO` enables (1) or disables (0) direct I/O for chunks.
`DIRECT_IO 1`
- IDS with no aging:
IDS chunks in file system and on raw devices
Pre-settings for the IDS configuration:
 - onconfig parameter `VPCLASS cpu` configures the CPU VPs.
`VPCLASS cpu,num=<#>[,max=<#>][,aff=<#>],noage`
- IDS with processor affinity:
IDS chunks in file system and on raw devices
For this test scenario, the VM was configured to have two CPUs. This is because testing processor affinity with just one CPU available in the VM does not make much sense.

For testing processor affinity, the configuration is a bit more complicated. The important settings are:

- VM re-configured with two CPUs.
With that the numbering of the CPUs can be seen by looking at "cpuinfo": `cat /proc/cpuinfo`.
- Pre-settings for the IDS configuration:
 - `MULTIPROCESSOR 1`
 - `SINGLE_CPU_VP 0`
 - `VPCLASS cpu,num=2[,max=<#>],aff=0-1[,noage]`

Test results

The test results verified that all the OS-specific functionality of IDS under test work correctly in a Xen hypervisor VM. No issues were detected during the test runs. The tested functionality of IDS can be used in a Xen hypervisor VM without restrictions.

Table 1. Test result summary

Configuration	With raw devices	With cooked files
IDS normal	OK	OK
IDS with direct I/O	OK	OK
IDS with no aging	OK	OK

IDS with processor affinity

OK

OK

A word on performance

As mentioned before, the purpose of this testing is to verify functionality -- not to measure or state any performance (and that means either relative or absolute numbers). To state any meaningful performance it is first necessary to seriously measure performance. And this is not the objective for this testing. Also, a different hardware setup with more resources (CPUs, memory, disk space, network bandwidth) would be needed. Having additional resources would allow the application (for example, TPC-x suite) to be off-loaded to one of several client machines. A larger IDS instance could run in a VM using more CPUs and memory, or several IDS instances could run in different VMs, each using its allotted share of the resources. After all this would be a realistic scenario of using virtualization for several IDS instances. Therefore such performance testing is reserved for a future project.

Nevertheless, as a byproduct of running the functional tests as described, it was observed that the runtimes between IDS in the VM and IDS in Dom0 differ very little. With the few tests run, it is not possible to determine a real trend. The differences observed could just as well be within the statistic standard deviation. At least this does not disprove the claim that Xen hypervisor provides for its VMs near-native performance.

Conclusion

Xen hypervisor on Linux offers advanced virtual machine management technology through its paravirtualization method. Because it is an open source technology, it is a low-cost system in comparison to other similar systems. Therefore, the low cost is expected to make the popularity of Xen hypervisor increase in the medium term and thus create a significant user base.

The OS-specific functionality of Informix Dynamic Server for Linux is fully operational in a Xen hypervisor managed virtual machine. The tests performed do not reveal any problems or issues. As shown here, Xen hypervisor on Linux is a viable option for virtual machine environments running IDS instances. Such scenarios include hardware consolidation, complex test setups requiring multiple Linux OS instances, or dedicated virtual machines for hosting services.

Although performance measurements are not the objective of this testing, it can be said that no significant performance difference is observed during this testing when comparing IDS runtime in a virtual machine versus runtime in the native Linux domain.

Resources

Learn

- [developerWorks Informix technical resource center](#): Find more resources for Informix developers.
- <http://www.xen.org> is the home page of the Xen Open Source community.

Get products and technologies

- [SUSE Linux Enterprise](#) internet page from Novell.
- Download a free trial version of [Informix Dynamic Server](#).

About the authors

Martin Fuerderer

Martin Fuerderer is a software developer who has been working on IBM Informix Database Server for the last nine years. He was the coach for Nicole Neubuerger while she was doing the project work described in this article.

Nicole Neubuerger

Nicole Neubuerger is a student at the University of Applied Sciences Munich, Germany. She studies Information Systems and as a practical assignment she did the research and tests that served as basis for this article.