The Xen Hypervisor and its IO Subsystem

virtualizing a machine near you

Muli Ben-Yehuda, Jon D. Mason

`muli@il.ibm.com, jdmason@us.ibm.com`

IBM Haifa Research Lab, IBM Linux Technology Center

Table of Contents

The Xen Hypervisor

Quick Overview

- Open source hypervisor (aka virtual machine monitor), licensed under the GPL
- Provides secure isolation, resource control and QoS
- Requires **minimal** operating systems changes, and no userspace changes
- Supports x86, x86-64, ia64 and PPC in varying degrees of maturity
- Supports Linux, NetBSD, FreeBSD, OpenSolaris, ...
- Close to native performance!
- Supports live migration of VMs
- Widespread hardware support, including direct access
- Xen 3.0.0 just released!

Quick Overview of Xen cont’

- Developed and maintained at the Cambridge University Systems Research Group, by Ian Pratt, Keir Fraser, lots of others.
- Contributions from Intel, AMD, HP, IBM, others.
- Commercial backing available from `http://www.xensource.com/`, others.
- Read about it: `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`
- Get the source: `http://xenbits.xensource.com/`

Bibliography

Papers and presentations are available from

`http://www.cl.cam.ac.uk/Research/SRG/netos/xen/architecture.html`

- Xen and the Art of Virtualization, Paul Barham et al, SOSP 2003
- Xen and the Art of Repeated Research, Brian Clark et al, FREENIX 2004
- Safe Hardware Access with the Xen Virtual Machine Monitor, Keir Fraser et al, OASIS ASPLOS 2004 workshop
- Live Migration of Virtual Machines, Christopher Clark et al, to be published at NSDI 2005

Full Virtualization

- Full virtualization refers to running an unmodified OS on a virtual machine (e.g. VMWare). There are many ways to do this - for example binary rewriting of the running OS image.
- Hardware support makes full virtualization much easier.
- x86 is notoriously difficult to virtualize because some privileged instructions simply fail silently, rather than raising a trap.
- Newer Intel CPUs include virtualization support (VMX, VTx, VTi). AMD's comparable Pacifica technology will debut in Q1 2006.

Para-Virtualization

Para-virtualization refers to modifying the OS to make virtualization faster - modifying the OS to run on the virtualized environment rather than on bare metal.

- Easy to do when you have the source
- Can be combined with full virtualization techniques - para-virtualize where you can, use full-virtualization techniques where you can’t avoid it.
- XenoLinux - a port of Linux to run under the Xen hypervisor.
- The bulk of the work is replacing privileged instructions (e.g. cli, hlt, write to cr3) with hypervisor calls.
- Core concept: modify the OS to the virtualized environment, but expose some details of the hardware for optimization.

Virtualizing IO

Instead of providing physical devices, provides virtualized views of them.

- The full-virtualization way: emulate real devices (many ways to do this, most common is a software implementation of the hardware state machine)
- The para-virtualization way: class drivers and devices

Why not expose physical devices? can't do it securely, and can't do sharing, unless the device knows how to do it (PCI-SIG IOV group is working on it)

Virtualizing IO - The Xen way

- When Xen boots up, it launches dom0, the first privileged domain - currently has to be Linux 2.6, but in theory can be any other OS that has been properly modified
- dom0 is a privileged domain that can touch all hardware in the system (long term goal is to move all hardware handling to dom0, we’re three quarters of the way there)
- dom0 exports some subset of the the devices in the system to the other domains, based on each domain’s configuration
- The devices are exported as “class devices”, e.g. a block device or a network device, not as a specific HW model.

Xen IO

Frontends and Backends

- dom0 runs the backend of the device, which is connected to each domain's frontend for that device
  - netback, netfront for network devices (NICs)
  - blockback, blockfront for block devices
- backends and frontends communicate at a high level device abstraction - block class, network class, etc. The domain doesn't care what kind of block device it's talking to, only that it looks like a block device.
- domains other than dom0 may be granted physical device access, securely [as secure as the architecture allows, anyway]. This used to work in 2.0 but is currently broken in 3.0 and the unstable tree

Frontends and Backends cont'

Ultimately, all communication between frontends and backends happens in memory. Xen provides several mechanisms to make life interesting - err, easy - for driver developers:

- shared memory, and
- producer consumer rings, together with
- virtual interrupts,
- give us event channels...
- ... but what about bulk data transfers? grant tables to the rescue!
- how do we we tie it all up together?

Driver Domains

- we already saw that Xen gives one domain, dom0, access to all HW devices and other domains perform IO through it
- what if we could have multiple dom0's? each with its own devices and its own consumers (other domains that are accessing the hardware through it)
- with Xen 2.0, we can. In a sense, we run multiple drivers each in its own domain - hence “driver domains”.
- driver domains require the ability to hide PCI devices from dom0 and expose them to other domains.
- unfortunately some of the changes to move hardware initialization out of the hypervisor to dom0 during the 3.0 development cycle broke this functionality, and it's waiting for a volunteer to fix it...

Direct Hardware Access

One of the main selling points of virtualization is machine consolidation. So let’s assume for a second that you put your database virtual machine and your web server virtual machine on the same physical machine. Your database needs fast disk access; your web server, fast network access.

Xen supports the ability to allocate different **physical** devices to different domains (multiple “driver domains”). However, due to architectural limitations of most PC hardware, this cannot be done securely. In effect, any domain that has direct hardware access has to be considered “trusted”.

The Problem with Direct Access

The reason why is that all IO is done in physical addresses. Consider the following case:

- domain A is mapped in 0-2GB of physical memory
- domain B is mapped in 2-4GB of physical memory
- domain A has direct access to a PCI NIC
- domain A programs the NIC to DMA in the **2-4GB** physical memory range, overwriting domain B's memory. Ooops!

The solution is a hardware unit known as an "IOMMU" (IO Memory Management Unit).

DMA

DMA - Overview

Direct memory access (DMA) is the hardware mechanism that allows peripheral components to transfer their I/O data directly to and from main memory without the need for the system processor to be involved in the transfer.

From "Linux Device Drivers, 2nd Edition" By Alessandro Rubini & Jonathan Corbet

DMA - How it works

- Device Driver programs the adapter with physical memory address to DMA to/from.
- Adapter performs DMA on given address
- Adapter signals device driver via interrupt that DMA is complete

DMA - Problems?

- DMA limited to address the device can address (ie, 32bit devices can only access 32bits of memory)
- Devices can DMA anywhere they want.....

IOMMUs

Introduction

An I/O Memory Management Unit (IOMMU) are hardware constructs that can be emulated in software.

IOMMUs provides two main functions: Translation and Device Isolation

- The IOMMU translates memory addresses from “IO space” to “physical space” to allow a particular device to access physical memory potentially out of its range. It does this translation by providing an "in range" address to the device and either translates the DMA access from the "in range" address to the physical memory address on the fly or copies the data to the physical memory address.
- Also, IOMMUs can limit the ability of devices to access memory, used for restricting DMA targets.

Why do we need an IOMMU?

- Pros
  - 32bit DMA capable, non-DAC, devices can access physical memory addresses higher than 4GB.
  - IOMMUs can be programmed so that the memory region appears to be contiguous to the device on the bus (SG coalescing).
  - Device Isolation and other RAS features.
- Cons
  - TANSTAAFL, "there ain't no such thing as a free lunch." Remapping adds a performance hit to the transfer (can be mitigated by a TLB).

The Main Advantage - Isolation

But, the one we care the most about for virtualization is **isolation.** For isolation, it is not enough to translate -

- we need a translation to be available to a given device, but not to some other device
- we also need to restrict which domain can program which device

Unfortunately, not many of today's IOMMUs can actually do isolation.

Types of IOMMUs

- Software
  - Linux's swiotlb
  - Xen's grant tables
- Hardware
  - AMD GART (translation only)
  - IBM TCEs (translation and isolation)
  - AMD Pacifica (translation and isolation)

swiotlb

- Linux includes **swiotlb** which is a software implementation of the translation function of an IOMMU. Or we can just call it “bounce buffers”.
- Linux always uses swiotlb on IA64 machines, which have no hardware IOMMU, and can use it on x86-64 when told to do so or when the machine has too much memory and not enough IOMMU.
- As of 3.0.0, Xen always uses swiotlb in dom0, since swiotlb provides machine contiguous chunks of memory (required for DMA) unlike the rest of the kernel memory allocation APIs when running under Xen.
- Using swiotlb (or any other IOMMU) is completely transparent to the drivers - everything is implemented in the architecture’s DMA mapping API implementation.

swiotlb WIP

- At the moment, Xen has its own hacked-up copy of swiotlb. We are working on patches to generalize x86-64's dma-mapping code and merge Xen's swiotlb back into the stock swiotlb.
- The swiotlb code is wasteful in memory, since it requires a large physically contiguous memory aperture for the bounce buffers. The size of the aperture is configurable and ranges from several to hundreds of megabytes.
- Implementing support for multiple apertures will alleviate the need for pre-allocating so much memory.

Grant Tables

- Grant tables are a way to share and transfer pages of data between domains. They give (or "grant") other domains access to pages in the system memory allocated to the local domain. These pages can be read, written, or exchanged (with the proper permission) for the purpose of providing a fast and secure method for domains to receive indirect access to hardware.
- They are faster because driver domains are able to DMA directly into pages in the local domain's memory, instead of having to DMA locally and copying or flipping the page to the domain. However, it is only possible to DMA into pages specified within the grant table.
- Of course, this is only significant for non-privileged domains (as privileged domains could always access the memory of non-privileged domains).

Grant Tables and IOMMUs

- Grant tables, like swiotlb, are a software implementation of certain IOMMU functionality. Much like how swiotlb provides the translation functionality of an IOMMU, grant tables provide the isolation and protection functionality. Together they provide (in software) a fully functional IOMMU.
- Why not allow hardware acceleration if it currently exists?
- It should be possible to replace certain one or both of them with their hardware accelerated counterpart. For example, replacing swiotlb with AMD GART for translation, and still having grant tables to provide the protection. Unfortunately, the current infrastructure in Xen does not apply itself well to this and will require a re-write.

How Grant Tables Work - Shared Pages

- A driver in the local domain's kernel will advertise a page to be shared (via the gnttab_grant_foreign_access system call). This call notifies the hypervisor that this page can be accessed by other domains. The local domain then passes a grant table reference ID to the remote domain it is "granting" access to this page. Once the remote domain is done, the local domain removes the grant (via the gnttab_end_foreign_access call).
- This is used by block devices (and any other device that receives data synchronously).

Transferred Pages

- Also known as "Page flipping"
- Used by network devices (and any other device that receives data asynchronously)
- A driver in the local domain's kernel will advertise a page to be transferred (via the gnttab_grant_foreign_transfer call). This call notifies the hypervisor that this page can be received by other domains. The local domain then transfers the page to the remote domain and takes a free page (via producer/consumer ring).

Shared vs. Transfer, Why the difference?

- Block devices already know which domain requested data to be DMA'ed.
- Incoming network packets need to be inspected before it can be transferred.
- Newer networking technology (such as RDMA NICs and Infiniband) have the ability to DMA directly into domUs, and will not need to transfer pages.

AMD GART

- AMD Graphical Aperture Remapping Table (GART) provides a basic, translation only, IOMMU
- Implimented in the on-chip memory controller
- Physical memory window and list of pages to be translated
- Addresses outside the window are not translated
- Fully supported in Linux; Xen support is WIP

Pacifica IOMMU

- Pacifica IOMMU also provides translation and isolation.
- Translation is done via the AMD GART and isolation is provided via the Pacifica northbridge. Northbridge verifies DMAs against the DEV (Device Exclusion Vector) defining the access permissions for the device. If it fails, it returns all 1s for a read or suppresses the store on a write and returns a Master Abort.
- Each "protection domain" has a device exclusion vector (DEV) that specifies the per-page access rights of the devices in that domain. Each bit in the DEV corresponds to one 4kB page in physical memory.
- DEV checks are applied before the addresses are translated via GART.
- Pacifica IOMMU is very similar to Calgary TCEs.

Calgary TCEs

- Calgary’s Translation Control Entry (TCE) provides functionality to translate and isolate
- Provides a Unique I/O Address space to all devices behind each PCI Host Bridge (PHB)
- Rather than allowing DMA devices to access memory directly, I/O translation uses DMA address as indexes into a system controlled translation table in memory. Anything not in this table can not be accessed. This gives it the ability to protect domains from errant or hostile DMA requests.
- Currently implemented in IBM’s pSeries servers.

Summary

- At the moment, Xen does not support any HW IOMMU.
- We are working on it!
- Due to Xen’s architecture, a large chunk of the work is actually in Linux’s dom0 kernel, e.g. to support multiple IOMMUs on x86-64 cleanly.

And we have a few open questions, too:

- How do we squeeze the most performance out of systems with IOMMUs? what are the right interfaces?
- How can we design better IOMMUs?

Questions?

Backup

Direct Hardware Access

One of the main selling points of virtualization is machine consolidation. So let’s assume for a second that you put your database virtual machine and your web server virtual machine on the same physical machine. Your database needs fast disk access; your web server, fast network access.

Xen supports the ability to allocate different **physical** devices to different domains (multiple “driver domains”). However, due to architectural limitations of most PC hardware, this cannot be done securely. In effect, any domain that has direct hardware access has to be considered “trusted”.

The Problem with Direct Access

The reason why is that all IO is done in physical addresses. Consider the following case:

- domain A is mapped in 0-2GB of physical memory
- domain B is mapped in 2-4GB of physical memory
- domain A has direct access to a PCI NIC
- domain A programs the NIC to DMA in the **2-4GB** physical memory range, overwriting domain B's memory. Ooops!

The solution is a hardware unit known as an "IOMMU" (IO Memory Management Unit).