Xen and the Art of Virtualization

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt & Andrew Warfield

Presented by Ankur Mishra
What I plan to address

• Motivations for Virtualization
• How Xen works
• Xen vs. Bare Hardware vs. Disco/VMWare
  – In this case it is sometimes easier to use VMWare to compare against because (Xen and VMware) they were both designed for the x86 architecture
• The future of Xen
Virtualization & the Challenges

• Speed & Performance
• Security
  – Resource Isolation
• Functionality

• Xen & its target
  – The authors came up with the design goal of being able to run 100 simultaneous virtual machine implementations with Binary Compatibility
Breaking it Down

• Virtualization (today) can be broken down into two main categories
  – Full Virtualization
    • This is the approach that Disco and VMWare uses
  – Paravirtualization
    • This is the approach that Xen uses
The Traditional Approach

• Traditional VMM (Virtual Machine Monitor) exposes its “hardware” as being functionally identical to the physical hardware
  – This approach can be difficult to implement (especially with x86 systems)
  – There are also situations where it is useful to provide real AND virtual resources (for example virtual and real timers)
• Under this model, the “guest” machine would not have access to this information
Xen’s Approach

• Instead of making the virtual machine 100% functionally identical to the bare hardware, Xen makes use of **Paravirtualization**

• Paravirtualization is a process where the guest operating system is modified to run in parallel with other modified systems, and is designed to execute on a virtual machine that has a ‘similar’ architecture to the underlying machine.

  – **Pros:**
    • Allows for improved performance

  – **Cons:**
    • The hosted operating system must undergo modification before it can be hosted by the Xen Hypervisor (this can be a bit of a challenge)
Xen’s Design Approach
Xen: Notation

- **Guest Operating System**
  - The OS software that Xen hosts

- **Domain**
  - The virtual machine within which a guest operating system executes

- **Guest OS’es and domains are analogous to a program and a process**

- **Hypervisor**
  - This is the instance of Xen that handles all of the low level functionality
Under the hood
(how does Xen perform its magic?)

- The Xen paper discusses the following areas
  - CPU
    - Virtualization of the CPU
    - CPU Scheduling
    - Time & Timers
  - Memory Management
    - Virtual Address Translation
    - Physical Memory
  - Device I/O
    - Network
    - Disk
  - Control Transfer
Xen and the CPU

• This undoubtedly where the most change is required by the guest OS
• Xen challenges the assumption that the OS is the most privileged entity
• Privileged instructions
  – These are paravirtualized by requiring them to be validated/executed within Xen
Xen and the CPU

• The x86 is less difficult than most systems to virtualize
  – This is due to the built in security levels build within the x86 (known as rings)
  – Most systems have the OS running on ring 0 (the most privileged)
  – Most user software runs on ring 3
  – Ring 1 & 2 generally are not used
• Xen uses this fact to modify the OS to execute on ring 1
Xen, Scheduling, and Timers

• Xen currently uses an algorithm called the Borrowed Virtual Time algorithm to schedule domains.
• This is important to mitigate the problem of one domain executing code that can adversely affect another domain.
• Xen also provides several different types of timers
  – Real Time (time that always advances regardless of the executing domain)
  – Virtual Time (time that only advances within the context of the domain)
  – Wall Clock Time (time that takes into account local offsets for time zone and DST)
Control transfer & Eventing

- Exceptions and Eventing
  - These include memory faults and software traps
  - These are generally virtualized through Xen’s event handler
  - Typically the two most frequent exceptions that occur (enough to effect performance)
    - System Calls
    - Page Faults
  - These are two examples of a ‘fast’ handler (one in which bypasses the hypervisor)
Paravirtualization of the MMU

Paravirtualization

Full Virtualization

Diagrams provided by a presentation from the Universität Karlsruhe
Xen and Virtual Memory

• When the guest OS requires a new page table, it allocates it from its own memory reservoir
  – After this it is registered with Xen
  – The OS then gives up all direct write privileges to the memory
  – All subsequent updates must be validated by Xen
  – Guest OS’s generally batch these update requests to spread the cost of calling the hypervisor
• Segmentation is virtualized in a similar way
Xen and Virtual Memory

• Xen uses a design where
  – Guest OS’s are responsible for allocation and managing hardware pages
  – Xen exists in a generally unused section at the top of every address space. This is to ensure that the Xen is never paged out

• This differs from the approach that Disco takes where the Disco VMM keeps a second level of indirection.
  – Essentially VMM within VMM
Memory Management

• As discussed in an earlier class Memory Management can be quite challenging

• Some key challenge points
  – x86 does not have a software managed TLB
  • Its TLB is not tagged, which means that the TLB must be flushed on a context switch
Xen and Device I/O/ Management

- Data I/O is transferred to and from domains via Xen through the use of a buffer descriptor ring.
  - This is a system that is based around a pair of producer consumer pointers, one set used within the guest OS, the other within the Hypervisor.
  - This allows for the decoupling of when data arrives/is accessed and the event notification.
Control of the Hypervisor

- Domain0 is given greater access to the hardware (and hypervisor). It has a guest OS running on top of it as well, but also has additional “supervisor” software to manage elements of the other existing domains.
- This is different than VMWare which has the notion of a Host OS acting underneath it.
Disk I/O (The Differences)

• Disco acts as the go between for Disk I/O
• Xen allows Domain0 to have direct access to the disk.
  – Domain0 houses virtual block device (VBD) management software
  – The VBD makes use of the ring mechanism
  – Subsequent domains confine their disk access through the VBD management software
  – This allows Xen to maintain a tighter control over disk access, and to allow “batching” of disk requests
• VMWare (from experience) allows for several options for Disk I/O.
  – To allow the guest OS unfettered access to the raw device—basically as a “pass through”
  – Allow VMWare to create a “virtual disk” that is a binary file that is contained within the file system of the host OS, and is controlled by the VM Virtual Machine
• These are also different from running an OS on top of bare hardware, where Disk I/O is managed by the OS
Building a new Domain on Xen

- Domain0 is a privileged domain
- New domain creation is delegated to Domain0
  - This offers the advantage of reducing the complexity of the hypervisor
  - Additionally building new domains that originate from Domain0 allow for a better debug environment
Networking

- Networking and Computers go hand in hand today
- Because of this, Xen also provides a “Virtual Firewall”
  - Domain0 is responsible for creating the firewall rules (can we see a common theme emerging?)
  - Data is transmitted (and received) using two buffer rings (one for outgoing, the other for incoming data)
  - Incoming data packets are analyzed by Xen against the Virtual Firewall rules, and if any are broken, the packet is dropped
Other Hardware

• What was observable from the block diagram for Xen was that you still have the notion of Xen enabled hardware drivers
• This is similar to how VMWare operates.
  – For instance if you have a sound card on your machine, the hosting guest machine will detect that you have a Sound Blaster enabled sound card.
  – Another example (from VMWare) is the video driver that can be installed on the guest OS to improve video performance.
• This is another hidden challenge of virtualization
  – Not only do you have to virtualize the memory and CPU, but also any other devices that the guest OS can access!
Evaluation

• Relative performance
  – Compared performance of three virtualization techniques with Native Linux

• Concurrent virtual machines
  – Compared performance of Xen with Native Linux
  – Compared performance of Xen by increasing number of OS instances
Relative Performance

Figure 3: Relative performance of native Linux (L), Xenolinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).
Concurrent Virtual Machines
Linux vs Xen

Aggregate number of concurrently spawning clients

Simultaneous SPEC WEB99 Instances on Linux (L) and Xen (X)
Conclusion

• Architectures such as x86 does not support full virtualization
• Xen is a high performance virtual machine monitor which uses Para virtualization
• Modification to the kernel code of guest OS is required
• Performance achievement near to that of Native Linux
Xen and the Future

• This paper was presented October 2003. Since then, the popularity of Xen has increased to include support from vendors such as
  – Sun Microsystems
  – Hewlett-Packard
  – Novell
  – Red Hat
  – Intel
  – Advanced Micro Devices
  – Voltaire
  – IBM
Xen and the Future

- To quote the news.com article (see my works cited list for the complete article)

  “The requirement for a modified operating system will loosen with Intel’s coming Vanderpool Technology” (Vanderpool is a hardware virtualization project)

- Additionally, AMD announced they are working on bringing Xen to their 64 bit platform
- Intel has experimental support on its Itanium chipset
- IBM is also working on a variant of the Hypervisor (a “Secure Hypervisor”) that adds more protections against attacks.
Works Cited

• I used several diagrams from the following paper for this presentation

• This link provided some up to date info on Xen

• Information about Vanderpool can be found here

• Some Slides are taken from previous Class
  - http://web.cecs.pdx.edu/~walpole/teaching.html