Xen and the Art of Virtualization

Introduction

- Three challenges listed; what are they? Does the paper address them?
    a. VMs must be isolated so that they can not interfere with each other. Especially performance isolation.
    b. There must be support for many different operating systems.
    c. Performance overhead must be small.

  - The main focus The main focus of Xen is on third point i.e. imposing little performance overhead
  - The paper didn't address supporting many OSes very thoroughly. They ported Linux and were in the process of porting Windows XP.

- What is the main point that the intro makes to motivate Xen?
  The authors looked at different ways to utilize hardware by running multiple applications on a single machine without virtualization, but those didn't provide performance isolation between applications. The paper is an exploration into the tradeoff between performance and heterogeneity. In other words, does the performance of paravirtualization outweigh the costs of having to alter every operating system in order to make it compatible with the hypervisor. While the authors don't specifically focus on this tradeoff, it is something to keep in mind.

- Xen Vs Denali
  (Some thoughts from the class discussion a. Denali is good from research point of view not for commercial point of view b. 100 full fledge OS is better than 10000)
  1. Denali does not target existing Application Binary Interfaces (ABI) so it can elide certain architectural features from their VM interface.
  2. In Denali, each VM essentially hosts a single-user single-application unprotected OS. (the reason is lack of virtual MMU)
  3. In Denali VMM performs all paging to and from disk, malicious virtual machines can encourage thrashing behavior. On the other hand In Xen, guest OS performs its own paging.
  4. In Denali, protection is achieved by separating the namespaces whereas Xen believes secure access control within the hypervisor is sufficient to ensure protection

- Denali targeted thousands of hosted OS instances; Xen 100. What is the usual limit to such instances? What is a realistic goal?
  While Denali focused on the extreme end of how many VMs can be squeezed onto a system, Xen focused more on making a usable system. Denali didn't really care about the usefulness of the operating system running on top of it where Xen specifically wanted full fledged systems to boot. Scalability can help determine what a realistic goal for supporting large numbers of instances. Things to consider include:
    a. Active instances vs sleeping instances

b. Hardware limitations
c. VMM limitations
d. Network Congestion (Lots of VMs may cause inability to communicate)
e. Workload

Approach and Overview

- What are the reasons listed for the partial virtualization Xen supports?
  Xen used paravirtualization because x86 was never meant to be virtualized and therefore didn't support it. The x86 architecture has instructions that are not technically privileged but still rely on the current mode of the processor. x86 has a hardware managed page table which makes virtualization more difficult. The hardware walks the TLB but does not do both virtual -> physical memory translations and physical ->machine memory translations. Therefore, it is difficult for the VMM to know when to get involved in all situations. There are also situations where it might be beneficial for the guest OS to be able to see the hardware. In order to do that it must be aware that it is running on a hypervisor. While the issues in virtualizing x86 are difficult, they are not impossible (ESX Server employed different solutions). One of the major benefits of paravirtualization is the simplicity it lends to the actual realization of a VMM.

- Four design principles are listed; what are they, and what do you think of them?
  a. Support for unmodified binaries
  b. Support full multi application OSes
  c. Paravirtualization for performance
  d. Hiding the effects of resource virtualization from guest OSes risks both correctness and performance. One example that was talked about was contiguous memory. An OS relies on contiguous memory in order to support superpages. If the OS is not aware that its memory is virtualized, it could cause correctness issues.

  - First two(1,2) focus on usability and last two (3,4) on performance.

- Claim is that with a software-managed TLB, virtualization is easier; why is this so? (What is hard about a hardware-managed approach?)
  - X86 supports H/W managed TLB's. Virtualizing S/W managed TLB is easier that H/W managed TLB because in H/W manages TLB control goes to processor which walk through page table in H/W and in S/W manages TLB, TLB miss causes a trap and which is easy to emulate.
  - In Xen guest OSes are responsible for allocating and managing the H/W page tables with minimal involvement from Xen.

- Why is it important that Xen is mapped into each VM address space?

By having Xen in each VM's address space it allows control to go in and out of the hypervisor without having to flush the TLB every time. This small change allowed a big performance gain.

- How is the Xen approach to guest page tables different than shadow page tables used by more traditional VMMs on x86?
  Since Xen doesn't use shadow pages, it can keep a single copy of the page table instead of two. Xen forces the guest OS to access pages through Xen by mapping the page table read only. Essentially Xen added an interface to access pages. Xen allowed for batching of updates to the page table in order to keep from taking a massive performance hit from calling into Xen for every change to the page table.

- Why are x86 rings important?
  Rings allow Xen to have the most privilege on the system while still making is possible for the guest OSes to have a higher privilege and remain isolated from their applications. This yields higher performance since there is no need for separate address spaces.
  - X86 supports 4 privilege levels called as rings (ring 0, 1, 2, 3) with ring 0 with highest privilege.
    - ring 0 -> Xen
    - ring 1 -> OS
    - ring 2 -> unused
    - ring 3 -> Applications

- The text says "A table describing the handling for each type of exception is registered with Xen"; how is this different than a typical VMM?
  A typical VMM has to intercept and "snoop" on the guest OS when it tries to set up its exception handlers on boot. In Xen, the guest OS registers its exception handlers directly with Xen. Xen also supports a "fast" exception handler for system calls. This eliminates the need for indirection via ring 0 on every exception. This can be done since Xen validates each "fast" handler by checking to make sure it can't run in ring 0.

- What then happens upon an exception?
  When an exception happens, a privileged register needs to be read. So, Xen must get involved. Xen makes a copy of the exception stack frame on the guest OS's stack (which includes the value of the register) and then invokes the OS's handler.

- Why must the page-fault handler be written differently than usual?
  As mentioned above, a privileged register must be read (CR2). This can only be done from ring 0 so Xen must read the register to determine the actual page that faulted. Then, Xen can pass that value into ring 1 when it copies the exception stack frame to the guest OS's stack.

- Domain0 Discussion
    a. Domain0 is created at boot time. Domain0 is responsible for hosting the application-level management softwares.
    b. Domain0 can use control interface and which has ability to create and terminate other domains and control their scheduling parameters.
    c. Is there is risk because of that ? may be or may not be :)

Detailed Design

- How come every paper claims to be working on some other OS port?
  It shows that porting an OS isn't an easy thing. Modifying a complex code base takes time.

- What is the main communication mechanism between OS and Xen?
  When the OS needs to communicate with Xen it uses a synchronous hypercall to call into Xen. When Xen needs to communicate with a guest OS it uses an asynchronous event system.

- How is time virtualized?
  There are three timers provided by Xen:
    a. Real Time (nanoseconds since boot)
    b. Virtual Time (advances while guest OS is running)
    c. Wall-Clock (offset to real time)

- For memory virtualization, how does Xen "validate" a PT update?
  First, Xen checks to make sure that the memory belongs to the OS making the update. Frames also have a mutually exclusive type. Page Directory (PD), Page Table (PT), local descriptor table (LDT), global descriptor table (GDT), or writable (RW). Therefore, if a frame is of type PT, an OS can not modify it directly since it can't be both PT and RW.

- OSes can batch PT updates for performance; what kind of race does this introduce?
  A race can happen if an OS has batched many PT updates but the updates haven't been processed by Xen yet. During the time between the batching and the actual execution of the update, if the OS needs a page updated in the batch, it won't be able to access it. The solution to this was for Xen to look for hints to process the batches. If an OS is about to flush the TLB then that is a logical time to process any outstanding batches. Nevertheless, the OS still has to make sure there are no outstanding updates when it incurs a fault.

- How is physical memory managed by Xen? By an OS on Xen?
  Xen has a shared translation array which is readable by all guest OSes. A guest OS gets its max amount of memory on creation. Then, Xen uses ballooning in order to aid in reclaiming memory from guest OSes. Otherwise, the guest OS request memory from Xen as well as gives memory back when it doesn't need it.

- Disks: why are reorder barriers needed?
  Reordering barriers are necessary for filesystem consistency. A filesystem writes checkpoints and data out to disk and if they are written in the incorrect order, then if the filesystem were to crash there would be the potential for inconsistency. Something to note, the OS is expecting specific behavior from the disk (in order writes), but the insertion of the VMM in between the OS and the disk violates these expectations. Thus, necessitating the need for reorder barriers. In general, whenever adding layers that break current expectations, it is necessary to provide upper layers the information about this change in behavior.

Evaluation

- The paper provides extensive evaluation results. The short of it is that Xen can host up to 100 virtual machines simultaneously on a server circa 2003, and Xen's performance overhead is only around 3% compared to the unvirtualized case.

Eval questions not covered in class.

- What is the experimental setup for this paper? Anything interesting about it?
- What does Figure 3 show us?
- What do we learn from Tables 3, 4, and 5?
- Do system calls execute quickly on Xen? Why?
- Why does fork take so long? Does the hypercall cost fully explain the difference?
- Why should micro benchmarks not be taken "too seriously"??
- What does the performance isolation experiment show us?
- What was the result of the "scalability" experiment?