INFLUENTIAL OS RESEARCH

Virtualization

Tobias Stumpf

SS 2014
Roadmap

Introduction
Fundamentals
TENEX
Xen and the Art of Virtualization
Binary Translation
FDBT
Conclusion
References
INTRODUCTION

History

- development start in the early 1960
- 1970 IBM System/370
- 1996 paravirtualization on top of a L4 kernel
- 1997 first version of Virtual PC for Macintosh
- 1997 Disco project
- 1998 VMWare was founded
- 2001 Connectix launches its first version of Virtual PC for Windows
- 2001 VMware launches its first virtualization solution for servers
- 2003 first release of Xen
**INTRODUCTION**

Virtualization

- abstraction layer between hardware and operating system
- virtualization gives a stronger isolation than an OS
  - using several OSes on the same hardware

<table>
<thead>
<tr>
<th></th>
<th>VM</th>
<th>VM</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMM</td>
<td>VMM</td>
<td>VMM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hypervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FUNDAMENTALS
Type I vs. Type II

Type I

VM
VM
VM
VMM
VMM
VMM
Hypervisor
Hardware

Type II

VM
VM
Application
Application
VMM
VMM
Hypervisor
Host OS
Hardware

IOS - Virtualization
FUNDAMENTALS
Requirements

defined by Popek & Goldberg [3]

**Equivalence** execution behavior of a virtual system should not differ from real hardware

**Resource control** the native system must be in complete control of all virtual resources.

**Efficiency** the virtual system should run a significant amount of time without intervention
**FUNDAMENTALS**

- **virtualization**
  - **emulation**
    - hardware emulation
    - p-code
  - **native**
    - full virtualization
    - para virtualization
      - classical
        - binary translation
          - hardware assisted virtualization
FUNDAMENTALS
Hardware Requirements

- different execution levels
- hardware has to trap, if a guest
  - executes a sensitive instruction
  - accesses a non-allocated resource
Problems with the x86-architecture

**Problems**

- some sensitive instructions do not trap
- OS can detect if it is not executed in ring 0
- x86 nearly fulfils virtualization requirements [4]

**Solutions**

- Binary Translation (VMWare Workstation, VirtualBox)
- Paravirtualization (Xen, L4Linux)
FUNDAMENTALS
Memory Virtualization

- Guest virtual address
- Guest page table
- Guest physical address
- Host virtual address
- Host page table
- Host physical address
- Shadow page table
TENEX
TENEX, a Paged Time Sharing System for the PDP-10

Daniel G. Bobrow, Jerry D. Burchfield, Daniel L. Murphy and Raymond S. Tomlinson
1. state of the art virtual machine
2. good human engineering throughout system
3. the system must be implementable, maintainable, and modifiable
• introduced in 1968
• word-size: 36-bit
• addresses-size: 18-bit
  → max. memory size 256 kilowords (1152 kilobytes)
• supervisor mode: I/O access, 1:1 memory translation
• user mode: memory partitioning (high / low memory address)
TENEX
BBN Pager

- interface between processor and memory bus (MMU)
- different mappings for user and monitor address space
- associative registers (TLB)
- 512 words per page
- 512 entries per page table
- pages can be shared between cores
new system call instruction (JSYS)
JSYS provides an independent transfer mechanism into the
monitor
transfer vector (one page), specified by the JSYS address
transfer vector defines where processor state is stored
methods to access user data within a monitor function
TENEX
TENEX Virtual Machine

- VM is a PDP-10 arithmetic processor with 256K of virtual memory
- paging hardware performs a trap on any data not in core
- core manager performs I/O
- traps are invisible for user processes
- a VM cannot use I/O instructions directly
- compatibility mode for DEC 10/50 binaries
TENEX

Jobs and Processes

- a job is a set of one or more hierarchically related processes
- communication by shared memory, software interrupts or direct control (only parent to child)
- use of processes: execute different programs / services, block for an event, debugging
- jobs are also used for accounting (who has to pay for execution time)
TENEX
File System

- uniform mechanism to communication with devices connected to the system
- read / write, write-only and read-only devices
- symbolic file name management
- file name (device name, directory name, file name, extension, and version number)
- access rights directory listing, read, write, execute, and append
- additional access rights for groups
CPU should be equally shared
administrator can prioritize a job
priority is given by \textit{runtime}/\textit{realtime}
priority boost for long blocked processes
Xen and the Art of Virtualization

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield
Xen and the Art of Virtualization

History

- started as paravirtualization hypervisor (University of Cambridge)
- first release and publication in 2003
- extended by Intel for hardware-assisted virtualization
- in 2004 Xen becomes commercial (XenSource Inc, today Citrix Systems)
- today Xen Project
XEN AND THE ART OF VIRTUALIZATION
Paravirtualization in a Nutshell

• guest OS knows that it runs in a VM
  – sensitive instructions are replaced by hypercalls
• guest OS acts like a user program
  – can inform the VMM if intervention is necessary (hypercalls)
• simplifies virtual memory management and access to I/O devices
**Xen and the Art of Virtualization**

**Structure**

- Introduction
- Xen: Approach & Overview
- Detailed Design
- Evaluation
- Related Work
- Discussion and Conclusion
Motivation

- support for performance isolation
  - existing systems lack on performance isolation
  - for closed groups not necessary
  - problem QoS crosstalk
  - isolation at lower levels can reduce the problem (e.g. Exokernel or Nemesis)

- Xen uses isolation below the OS layer
  - DRAWBACK: multiplexing whole OSes is more heavyweight

- live migration
XEN AND THE ART OF VIRTUALIZATION
Reasons against full virtualization

- distinguish from VMWare
- different virtualization problems with x86
- efficient virtualization is difficult (e.g. MMU)
- binary translation is expensive
XEN AND THE ART OF VIRTUALIZATION
Design Principles

1. support for unmodified application binaries
2. supporting full multi-application operating systems is important
3. paravirtualization for high performance and strong resource isolation
4. non-complete hiding of the effects of resource virtualization from guest OSes
XEN AND THE ART OF VIRTUALIZATION

Overview

Source: [2]
Memory management

- critical part
- software TLBs
- page tables are managed by the guest OS (hypervisor has to validate writes)
- highest 64 MB reserved for Xen
- segmentation
  - guest always fewer rights than Xen
  - guest cannot delete hypervisor entries
XEN AND THE ART OF VIRTUALIZATION
The Virtual Machine Interface

CPU
- OS assumes that it runs at the highest privilege level
- running OS and application at the same privilege level is expensive
- for x86, guest OS can run in ring 1

Device I/O
- no emulation to reduce overhead
- clean and simple interface for virtual devices
  (goals: efficient virtualization and isolation)
XEN AND THE ART OF VIRTUALIZATION

Costs of Porting

- port x86 architecture specific code to Xen’s interface
  - replace privilege instructions by hypercalls
  - remove most low level initialization code
- for WinXP even for architecture independent code
- Linux needs fewer changes than WinXP
- no information for NetBSD and WinXP virtual I/O drivers
Xen and the Art of Virtualization
Control and Management

• goal: separate policy from mechanism wherever possible
• hypervisor provides basic control operation
• specific control interface used by one authorized domain (the control VM)
  – create new VM
  – create / delete VIFs and VBDs
Hypercall: synchronous call from a domain into the hypervisor
Notification: notifying the domain from the hypervisor using asynchronous events
XEN AND THE ART OF VIRTUALIZATION
Data Exchange

Request Consumer
Private pointer in Xen

Request Producer
Shared pointer updated by guest OS

Response Producer
Shared pointer updated by Xen

Response Consumer
Private pointer in guest OS

- **Request queue** - Descriptors queued by the VM but not yet accepted by Xen
- **Outstanding descriptors** - Descriptor slots awaiting a response from Xen
- **Response queue** - Descriptors returned by Xen in response to serviced requests
- **Unused descriptors**

Source: [2]
XEN AND THE ART OF VIRTUALIZATION

Time / Scheduling

Scheduling

- Borrow Virtual Time scheduling algorithm
- work-conserving
- low-latency wake-up

Time

- **real time** ns since machine boot
- **virtual time** runs only during a VM is executed
- **wall clock** absolute time
Xen and the Art of Virtualization

Physical Memory

- expand / shrink during runtime
- balloon driver for guest
- guest OS expects continuous physical memory
  - Xen gives no guaranty
  - guest OS has to create the illusion of contiguous physical memory by itself
Xen and the Art of Virtualization

Network

- virtual firewall-router
- configures from dom0
- at least one interface for each VM
- virtual interface looks like real hardware
- two ring buffer (receive and transmit)
- network packages are not copied (scatter-gather DMA)
- round-robin packet scheduler
- Xen filters incoming packages
- a package with unknown receive is dropped
only dom0 has direct disk access
other domains use VBDs
DMA for zero-copy data transfer
Xen can reload write access
XEN AND THE ART OF VIRTUALIZATION

Evaluation

Benchmarks

- against other implementations
- against native performance

Environment

- Xeon 2.4 GHz dual core with 2 GB RAM
- Tigon 3 Gigabit Ethernet NIC
- 146 GB 10k RPM SCSI disk
XEN AND THE ART OF VIRTUALIZATION
Evaluation - Relative Performance
XEN AND THE ART OF VIRTUALIZATION
Evaluation - OS Benchmarks

IOS - Virtualization
XEN AND THE ART OF VIRTUALIZATION
Evaluation - OS Benchmarks

- forkProc
- execProc
- shProc

times in ms

L-SMP  L-UP  Xen

IOS - Virtualization
Xen and the Art of Virtualization
Evaluation - Context Switch

<table>
<thead>
<tr>
<th>Times in µs</th>
<th>L-SMP</th>
<th>L-UP</th>
<th>Xen</th>
<th>VMWare</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>2p0K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2p16K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2p64K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8p16K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8p64K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16p16K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16p64K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IOS - Virtualization

slide 42
Xen and the Art of Virtualization

Evaluation - Network

- Linux
- Xen
- VMWare
- UML

Bandwidth in Mb/s

TX (1500)  RX (1500)  TX (500)  RX (500)
XEN AND THE ART OF VIRTUALIZATION
Evaluation - Concurrent Virtual Machines

Aggregate number of conforming clients

Simultaneous SPEC WEB99 Instances on Linux (L) and Xen(X)
**Binary Translation**

Overview

- two types: static and dynamic
- translation between different instruction sets
- for x86 translation from x86 ISA to a x86 ISA-subset
- Usage: virtualization, sandboxing, debugging, testing, ...
most instructions can be simple executed (no translation)
jmp / branch or call / ret have to be modified
all non-trapping sensitive instructions, SGDT, SIDT, SLDT, SMSW, PUSHF, POPF
**BINARY TRANSLATION**

Example [1]

```c
int isPrime(int a) {
    for (int i = 2; i < a; i++) {
        if (a % i == 0) return 0;
    }
    return 1;
}
```
**Binary Translation**

Example [1]

```assembly
isPrime:
    mov  %ecx, %edi       ; %ecx = %edi (a)
    mov  %esi, $2         ; i = 2
    cmp  %esi, %ecx       ; is i >= a?
    jge  prime; jump if yes

nexti:
    mov  %eax, %ecx       ; set %eax = a
    cdq
    idiv  %esi        ; a % i
    test %edx, %edx    ; is remainder zero?
    jz   notPrim; jump if yes
    inc  %esi         ; i++
    cmp  %esi, %ecx    ; is i >= a?
    jl   nexti; jump if no

prime:
    mov  %eax, $1        ; return value in %eax
    ret

notPrime:
    xor  %eax, %eax      ; %eax = 0
    ret
```

IOS - Virtualization slide 48
Binary Translation
Example [1]

isPrime:

- `mov %ecx, %edi`
- `mov %esi, $2`
- `cmp %esi, %ecx`
- `jge prime`
**Binary Translation**

Example [1]

```assembly
isPrime:  mov  %ecx, %edi
          mov  %esi, $2
          cmp  %esi, %ecx
          jge  prime

isPrime': mov  %ecx, %edi ; IDENT
          mov  %esi, $2
          cmp  %esi, %ecx
          jge  [takenAddr] ; JCC
          jmp  [fallthrAddr]
```
**Binary Translation**

Example [1]

```assembly
isPrime:  *mov  %ecx, %edi    ; IDENT
         mov  %esi, $2
         cmp  %esi, %ecx
         jge  [takenAddr] ; JCC
         ; fall-thru into next CCF
nexti:   *mov  %eax, %ecx    ; IDENT
         cdq
         idiv  %esi
         test  %edx, %edx
         jz    notPrime  ; JCC
         ; fall-thru into next CCF
         *inc  %esi
         ; IDENT
```

IOS - Virtualization
FDBT
Fast Dynamic Binary Translation for the Kernel

Piyus Kedia and Sorav Bansal
Overview

- translator is implemented as loadable module for the Linux kernel
- transparent for the kernel
- transparency is sometimes relaxed for better performance
FDBT
Kernel Entry

State-of-the-Art

- replace entry point with a call to the DBT dispatcher (time intensive)

Improvement

- kernel entry points are directly exchanged with the translated counterparts (faster kernel entry)
State-of-the-Art

- existing systems provide a precise and consistent exception handling (e.g. DRK, VMWare)

Alternative

- give up preciseness
- benefit: faster and simpler to implement
- drawback: non-transparent
FDBT
Faster Design (Don’t care!)

- a PC value on the interrupt stack doesn’t have to represent a valid address
- interrupt handler can see the state between the translation of two native instructions
- translation addresses could be visible in the guest kernel
- idendity translation for call / ret instructions
- no interrupts and exceptions during the execution of the dispatcher
FDBT Evaluation

- 2x6 Intel Xeon X5650 2.67 GHz SMP with 4GB
- How to compare?
  - DRK not available
  - VMWare has more virtualization features
- use same workload as in the DRK evaluation
FDBT

Evaluation - Linux Build Time (scalability)

IOS - Virtualization
FDBT
Evaluation - Apache

Source: [2]
FDBT
Discussion

- outperforms existing solutions (VMWare, DRK)
- performance benefits come with function constrains
  - knowledge of the guest OS is required
- benefit of FDBT
  - can instrument all device drivers
CONCLUSION

- old and common technique
- different techniques (e.g. paravirtualization, binary translation)
- tradeoff between different requirements
REFERENCES


Xen and the art of virtualization.
REFERENCES

Formal requirements for virtualizable third generation architectures.

Analysis of the intel pentium’s ability to support a secure virtual machine monitor.

Scale and performance in the denali isolation kernel.