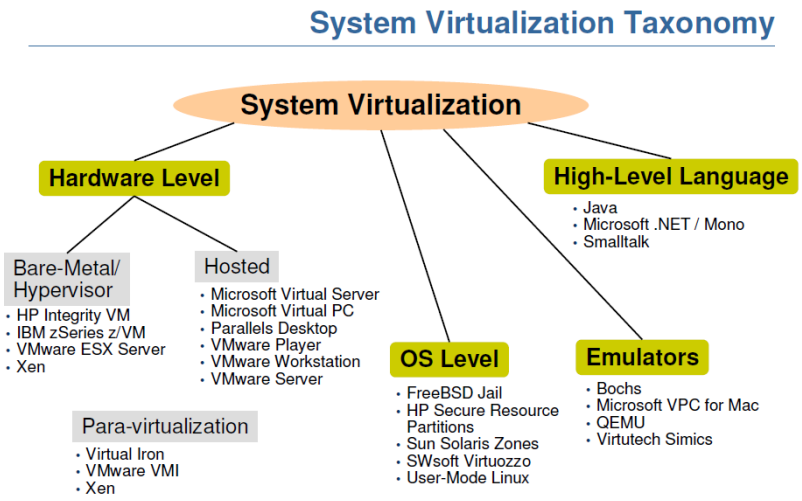
Revision History

Version	Date	Author	Description
0.1	04/24/2012	Hill Zhao	Initial draft.

Acronym and Codename Table

1 Foreword

2 Virtualization



3 ESXi Architecture

1 KeyFeatures

- Record-setting performance with up to 8,900database transactions per second, 200,000 I/O operations per second, and up to 16,000Exchange mailboxes on a single physical host
- Up to eight-way virtual SMP (symmetricmultiprocessing), enabling the virtualization of multiprocessor workloads • Memory over committment and deduplication, allowing higher consolidation ratios
- Broadest OS support of any hypervisor, enabling IT to virtualize numerous versions of Windows?, Linux®, Solaris®, NetWare®, and other operating systems.
- Built-in high availability through NIC teaming and HBA multipathing to protect against hardware component failures
- Up to 64 logical processing cores, 256 virtual CPUs, and 1TB RAM per host, enabling higher consolidation ratios

More refer: VMware-ESX-and-VMware-ESXi-DS-EN.pdf

2 Architecture

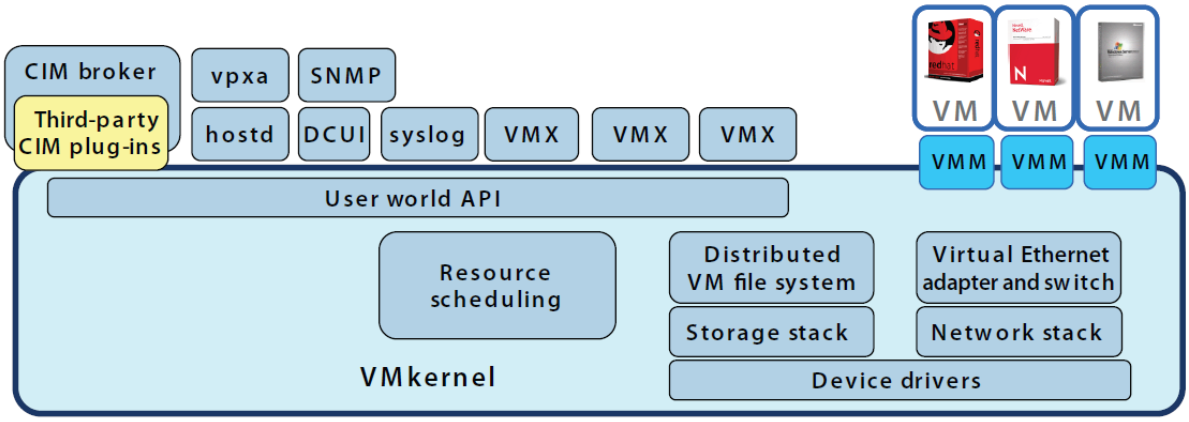
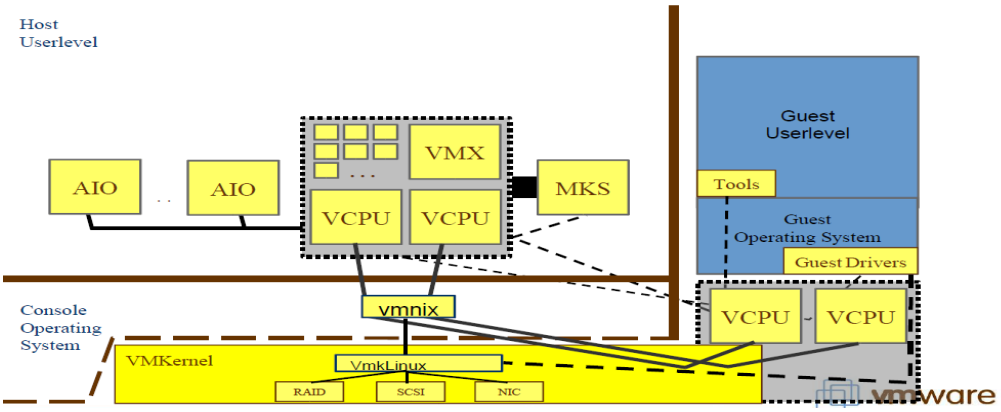
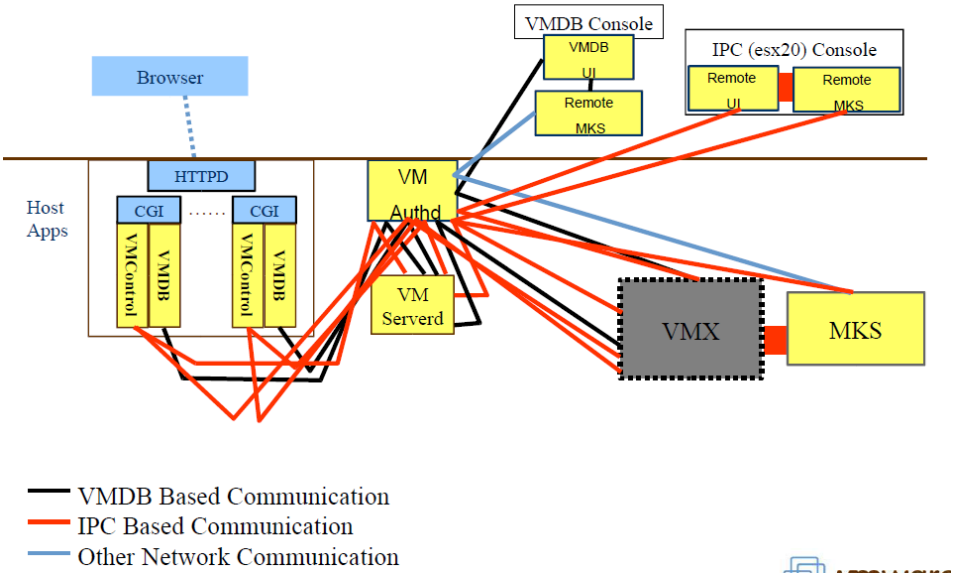


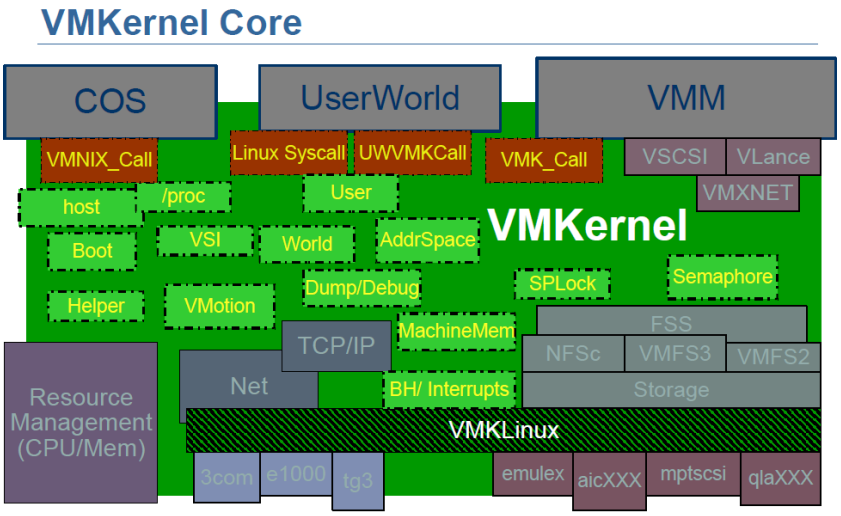
Figure 1: The streamlined architecture of VMware ESXi eliminates the need for a service console.

ESX Platform Boxes…



Connecting Outside the Box…





- Direct Console User Interface (DCUI) — the low-level configuration and management interface, accessible through the console of the server, used primarily for initial basic configuration.
- The virtual machine monitor, which is the process that provides the execution environment for a virtual machine, as well as a helper process known as VMX. Each running virtual machine has its own VMM and VMX process.
- Various agents used to enable high-level VMware Infrastructure management from remote applications.
- The Common Information Model (CIM) system: CIM is the interface that enables hardware-level management from remote applications via a set of standard APIs.

Reference:

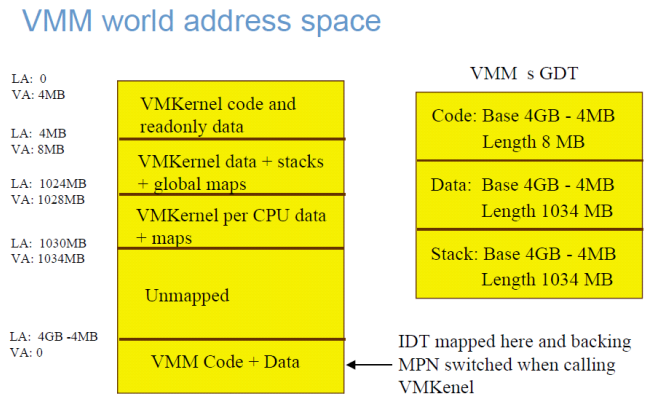
http://mylearn.vmware.com/mgrReg/registerComplete.cfm?id=10601673&operator=875284&ui=vmweb_rand

Why VMKernel?

No control on resource management
- CPU, Memory, Network, Disk
- Between VMs
- Between VM and host OS
  - Limiting host resources at boot time -> host won't provide functionality for resources host can't see.

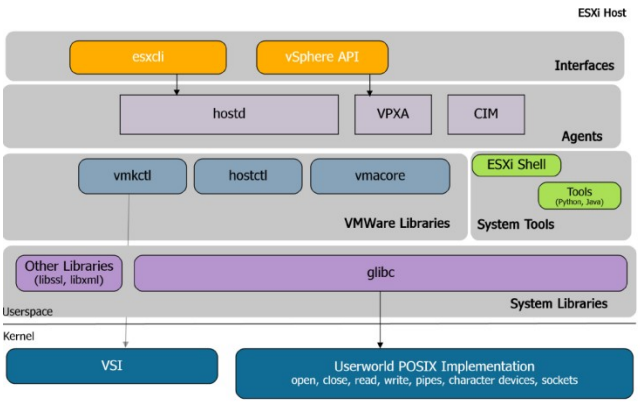
Limited host OS functionality
- NUMA, number of LUNs, SAN failover, interrupt balancing, etc...

Too many context switches



Reference:

vmk2003_2003.pdf



3 VMkernel

VMkernel is a POSIX-like operating system developed by VMware and provides certain functionality similar to that found in other operating systems, such as process creation and control, signals, file system, and process threads. It is designed specifically to support running multiple virtual machines and provides such core functionality as:

- Resource scheduling
- I/O stacks
- Device drivers

- Some of the more pertinent aspects of the VMkernel are presented in the following sections.

3.1.1 Boot & Log

3.1.2 File System

VMkernel uses a simple in-memory file system to hold the ESXi configuration files, log files, and staged patches. For familiarity, the structure of the file system is designed to be the same as that used in the service console of ESX. For example, ESXi configuration files are found in /etc/vmware and log files are found in /var/log/vmware. Staged patches are uploaded to /tmp.

This file system is independent of the VMware VMFS file system used to store virtual machines. Just as with ESX, a VMware VMFS datastore may be created on a local disk in the host system or on shared storage. If the only VMFS datastores used by the host are on external shared storage, the ESXi system does not actually require a local hard drive. By running diskless setups, you can increase reliability by avoiding hard drive failures and reduce power and cooling consumption.

Remote command line interfaces provide file management capabilities for both the in-memory file system and the VMware VMFS datastores. Access to the file system is implemented via HTTPS get and put Access is authenticated via users and groups configured locally on the server and is controlled by local privileges.

Because the in-memory file system does not persist when the power is shut down, log files do not survive a reboot. ESXi has the ability to configure a remote syslog server, enabling you to save all log information on an external system.

File system

VMFS version 2
Small number of large files
At least 1MB block size => small and fast metadata
Distributed (on-disk per file lock)
Single volume can span multiple disks
No caching in vmkernel for fast VMM access
COS /vmfs cache
COW/REDO logs
1 sector granularity, no synchronous parent read on writes

3.1.3 Users and Groups

Users and groups can be defined locally on the ESXi system. They provide a way to distinguish users that access the system via the Virtual Infrastructure Client, the remote command line interfaces, or the VIM API.

Groups can be used to combine multiple users, just as in other operating systems. Groups can be used, for example, to set privileges for many users at once. There are a few system users and groups, which are predefined in order to identify certain processes running in the VMkernel.

Administrative privileges can be set individually for each user or group. User and group definitions are stored on the file system in the files /etc/passwd, /etc/shadow, and /etc/group, and as in other operating systems, passwords are generated using standard crypt functions.

3.1.4 User Worlds

vmkernel

- The vmkernel is a small kernel that manages the machine
  - IDT and GDT
  - Interrupt controllers (PIC, APIC, IOAPIC)
  - Processors
  - Memory
  - Network and SCSI controllers
  - Timers
  - Disks including own file system

World Management

- One idle world is created for each CPU
- One helper world is created to perform asynchronous tasks
- One world is created per VMM
  - World is created by VMX
  - Address space initialized by VMX via COS/vmkernel calls
  - When ready, VMX calls vmkernel to start world running
  - Worlds destroyed by VMX on normal exit or world will kill itself if necessary
  - VMM worlds can run while VMX is running.

Remote Procedure Call

- Vmkernel provides remote procedure call (RPC) mechanism to communicate between worlds
  - Named channel
  - Blocking and non-blocking RPCs
  - Used by VMM world to communicate with VMX process.

PRDA

- Each world has a per-processor private data area (PRDA) mapped at a fixed location
- Contains CPU specific information
  - Scheduling info
  - Bottom-half handler info
  - Interprocessor interrupt (IPI) info

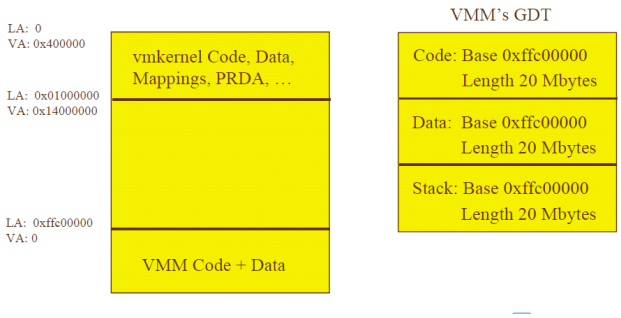
vmkernel and VMM interactions

- VMM lives in its own world
  - Takes over IDT and GDT
- VMM calls the vmkernel to perform certain actions
  - Interrupt forwarding
  - Network and SCSI device accesses
  - Memory management
  - RPC to VMX
  - Idle time handling
- vmkernel action queue that is processed by the VMM
  - Usually to indicate device status

Accessing the vmkernel

- The VMM calls the vmkernel via a function call, not a trap
- vmkernel must be mapped into the VMM address space
- Can't always be there unless take more of VM's address space
  - We tried this. It caused Windows 2000 performance to tank.
- Solution: map in vmkernel on demand
  - For minimal overhead we use segment wrapping
  - vmkernel starts after the VMM
  - GDT length setup to allow the vmkernel to come and go
  - Map in/out requires switching CR3.

Mapping in the vmkernel



Seeing Red: Lines of Communication

- IPC: Every which way at user-level, even remote.
- UserRPC: VMM -> VMX synchronous
- Monitor Actions: VMX / MKS / VMM -> VMM asynchronous
- VMKernel calls: VMM -> VMKernel synchronous
- VMKernel Actions: VMKernel -> VMM asynchronous
- Shared memory: All over among AIO <-> VMX, VMX <-> VMM, MKS <-> VMM, and more.

Seeing Red: Lines of Communication

- IPC: Every which way at user-level, even remote. (dying)
- VMDB: IPC replacement. Mostly outside the core platform, but fingers into the VMX and MKS.
- UserRPC: VMM -> VMX synchronous
- Monitor Actions: VMX / MKS / VMM -> VMM asynchronous
- Channels: Modular packaging of UserRPCs and Monitor Actions.
- VMKernel calls: VMM -> VMKernel synchronous
- VMKernel Actions: VMKernel -> VMM asynchronous
- Shared memory: All over among AIO <-> VMX, VMX <-> VMM, MKS <-> VMM, and more.

Reference:

VMkernel_2001.pdf

architecture_2001.pdf

architecture_2003.pdf

The term “user world” refers to a process running in the VMkernel operating system. The environment in which a user world runs is limited compared to what would be found in a general-purpose POSIX-compliant operating system such as Linux. For example:

- The set of available signals is limited.
- The system API is a subset of POSIX.
- The /proc file system is very limited.
- A single swap file is available for all user world processes. If a local disk exists, the swap file is created automatically in a small VFAT partition. Otherwise, the user is free to set up a swap file on one of the attached VMFS datastores.

3.1.4.1 Other User World Processes

Agents used by VMware to implement certain management capabilities have been ported from running in the service console to running in user worlds.

• The hostd process provides a programmatic interface to VMkernel and is used by direct VI Client connections as well as the VI API. It is the process that authenticates users and keeps track of which users and groups have which privileges. It also allows you to create and manage local users.

• The vpxa process is the agent used to connect to VirtualCenter. It runs as a special system user called vpxuser. It acts as the intermediary between the hostd agent and VirtualCenter.

• The agent used to provide VMware HA capabilities has also been ported from running in the service console to running in its own user world.

• A syslog daemon also runs as a user world. If you enable remote logging, that daemon forwards all the logs to the remote target in addition to putting them in local files.

• A process that handles initial discovery of an iSCSI target, after which point all iSCSI traffic is handled by the VMkernel, just as it handles any other device driver. Note that the iSCSI network interface is the same as the main VMkernel network interface.

In addition, ESXi has processes that enable NTP-based time synchronization and SNMP monitoring.

3.1.5 Direct Console User Interface

The Direct Console User Interface (DCUI) is the local user interface that is displayed only on the console of an ESXi system. It provides a BIOS-like, menu-driven interface for interacting with the system. Its main purpose is initial configuration and troubleshooting. One of the system users defined in VMkernel is dcui, which is used by the DCUI process to identify itself when communicating with other components in the system.

The DCUI configuration tasks include:

• Set administrative password

• Configure networking, if not done automatically with DHCP

Troubleshooting tasks include:

• Perform simple network tests

• View logs

• Restart agents

• Restore defaults

3.1.6 Open Network Ports

A limited number of network ports are open on ESXi. The most important ports and services are the following:

• 80 — This port serves a reverse proxy that is open only to display a static Web page that you see when browsing to the server. Otherwise, this port redirects all traffic to port 443 to provide SSL-encrypted communications to the ESXi host.

• 443 (reverse proxy) — This port also acts as a reverse proxy to a number of services to provide SSL-encrypted communication to these services. The services include the VMware Virtual Infrastructure API (VI API), which provides access to the RCLIs, VI Client, VirtualCenter Server, and the SDK.

• 427 (service location protocol) — This port provides access for the service location protocol, a generic protocol to search for the VI API.

• 5989 — This port is open for the CIM server, which is an interface for Third-party management tools.

• 902 — This port is open to support the older VIM API, specifically the older versions of the VI Client and VirtualCenter.

3.1.7 System Image Design

ESXi is designed for distribution in various formats, including directly embedded in the firmware of a server or as software to be installed on a server's boot disk. Figure 2 shows a diagram of the contents of the ESXi system image. Regardless of whether the image exists on flash memory or on the hard drive of a computer, the same components are present:

• A 4MB bootloader partition, which runs upon system boot up.

• A 48MB boot bank, which contains the 32MB core hypervisor code, along with a second alternate boot bank of the same size. The reason for two boot banks is explained below.

• A 540MB store partition, which holds various utilities, such as the VI Client and VMware Tools images.

• A 110MB core dump partition, which is normally empty but which can hold diagnostic information in case of a system problem.

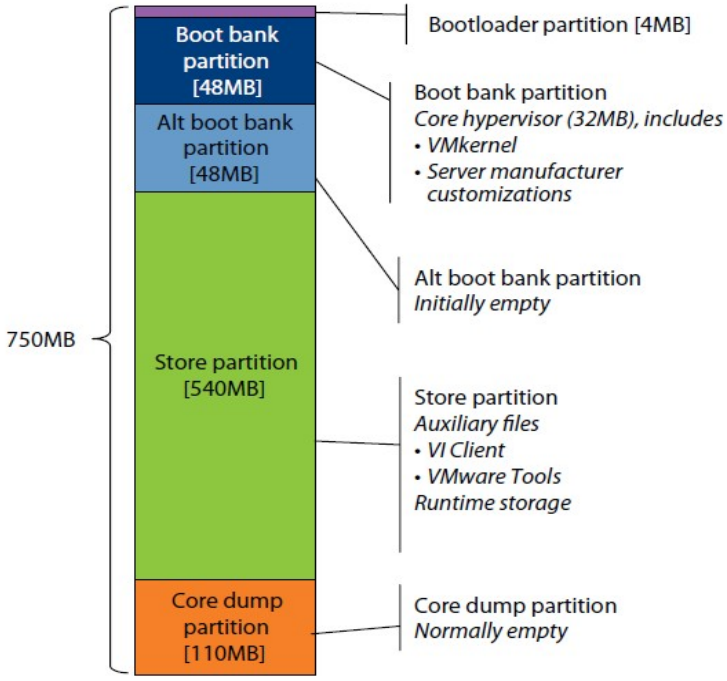


Figure 2: Contents of the ESXi system image

3.1.8 **Management Model for ESXi**

3.1.8.1 State Information

The state of an ESXi system is fully described by a handful of configuration files. These files control such functions as configuration of virtual networking and storage, SSL keys, server network settings, and local user information Although these configuration files are all found in the in-memory file system, they are also periodically copied to persistent storage

3.1.8.2 Common Information Model

The Common Information Model (CIM) is an open standard that defines how computing resources can be represented and managed. It enables a framework for agentless, standards-based monitoring of hardware resources for ESXi. This framework consists of a CIM object manager, often called a CIM broker, and a set of CIM providers.

CIM providers are used as the mechanism to provide management access to device drivers and underlying hardware. Hardware vendors, which include both the server manufacturers and specific hardware device vendors, can write providers to provide monitoring and management of their particular devices.

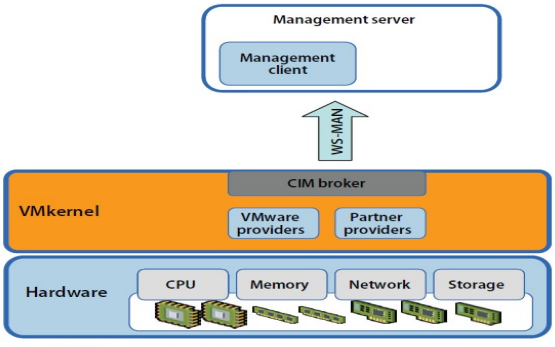


Figure 3: CIM Management model

3.1.8.3 VI API

The VMware Virtual Infrastructure API provides a powerful interface for developing applications to integrate with the VMware Infrastructure. The VI API enables your program or framework to invoke VirtualCenter Web Service interface functions on

VirtualCenter to manage and control ESX/ESXi. The VI SDK provides developers with a full environment for creating applications that interact with ESXi in a variety of programming languages

Reference:

ESXi_architecture.pdf

3.1.9 **CPU**

Conventional Schedulers

- Dynamic priority schedulers
  - Dominant paradigm: Unix, Windows, etc.
  - Complex “magic” parameters
    e.g., exponential decay rates
  - Crude non-linear control
    e.g., Unix “nice” values
  - No modularity
- Real-time schedulers
  - Restrictive assumptions
  - Disallow overcommitment
  - No modularity
- Other *ad hoc* schemes

Virtual Time Algorithms

- Basic algorithm
  - Associate virtual clock with each VM
  - Ticks at rate inversely proportional to VM shares
  - Allocation: select VM with minimum virtual time
- Example: 3 VMs A, B, C with 3 : 2 : 1 share ratio

A	2	4	4	6	8	8	8	10	10	
B	3	3	6	6	6	9	9	9	12	
C	6	6	6	6	6	6	12	12	12	

Reference:

ESXCPU_2001.pdf

3.1.9.1 Scheduler

Co-Scheduling SMP VMs

- Run all VCPUs of VM concurrently
  - May be required for correctness
  - Synchronous execution improves performance
  - Single vtime, stride for multi-VCPU VM
- Co-schedule
  - One VCPU scheduled by local processor
  - Map remaining VCPUs to remotely-preemptible processors
  - Send IPIs to force remote reschedules
  - Remote scheduler switches to specified VCPU
- Co-deschedule
  - Set timeout when local VCPU blocks
  - If still blocked, change remote VCPU states, send IPIs

Reference:

esx2-cpu-talk-final_2002.pdf

3.1.10 **Memory**

3.1.10.1 Reclamation Mechanisms

VMware Memory Management

Reclamation mechanisms

- Ballooning – guest driver allocates pinned PPNs, hypervisor deallocates backing MPNs
- Swapping – hypervisor transparently pages out PPNs, paged in on demand
- Page sharing – hypervisor identifies identical PPNs based on content, maps to same MPN copy-on-write

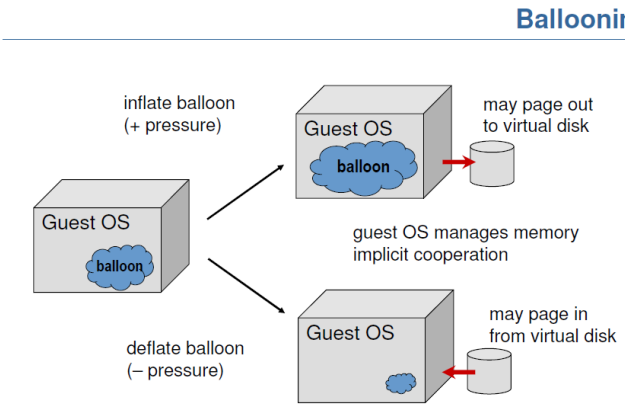
Allocation policies

- Proportional sharing – revoke memory from VM with minimum shares-per-page ratio
- Idle memory tax – charge VM more for idle pages than for active pages to prevent unproductive hoarding

- Key features
  - Flexible dynamic partitioning
  - Efficient support for overcommitted workloads
- Novel mechanisms
  - Ballooning leverages guest OS algorithms
  - Content-based page sharing
  - Statistical working-set estimation
- Integrated policies
  - Proportional-sharing with idle memory tax
  - Dynamic reallocation

3.1.10.1.1 Page Replacement Issues1

3.1.10.1.2 Ballooning



Ballooning Details

- Guest drivers
  - Inflate: allocate pinned PPNs; backing MPNs reclaimed
  - Use standard Windows/Linux/BSD kernel APIs
  - Related: Nemesis "self-paging" [Hand '99], Collective [Sapuntzakis '02]
- Performance benchmark
  - Linux VM, memory-intensive dbench workload
  - Compare 256 MB with balloon sizes 32 – 128 MB vs. static VMs
  - Overhead 1.4% – 4.4%
- Some limitations

3.1.10.1.3 Demand Paging

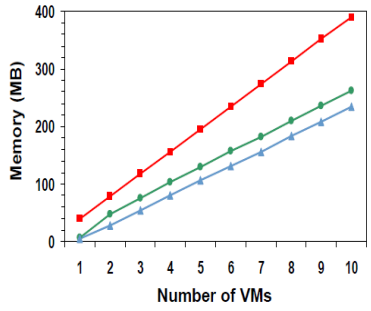
3.1.10.1.4 Sharing Memory

Sharing Memory

- Motivation
  - Multiple VMs running same OS, apps
  - Collapse redundant copies of code, data, zeros
- Transparent page sharing
  - Map multiple PPNs to single MPN copy-on-write
  - Pioneered by Disco [Bugnion '97], but required guest OS hooks
- New twist: content-based sharing
  - General-purpose, no guest OS changes
  - Background activity saves memory over time

3.1.10.2 Sharing Memory

Page Sharing Performance

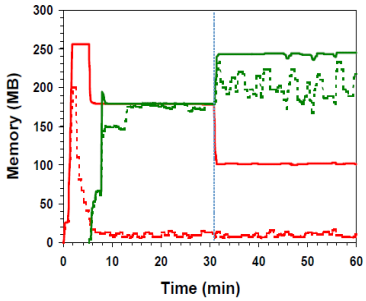


- "Best-case" workload
  - Identical Linux VMs
  - SPEC95 benchmarks
  - Lots of potential sharing
- Metrics
  - Total guest PPNs
  - Shared PPNs → 67%
  - Saved MPNs → 60%
- Effective sharing
- Negligible overhead

Reclaiming Idle Memory

- Tax on idle memory
  - Charge more for idle page than active page
  - Idle-adjusted shares-per-page ratio
- Tax rate
  - Explicit administrative parameter
  - 0% ≈ "plutocracy" ... 100% ≈ "socialism"
- High default rate
  - Reclaim most idle memory
  - Some buffer against rapid working-set increases

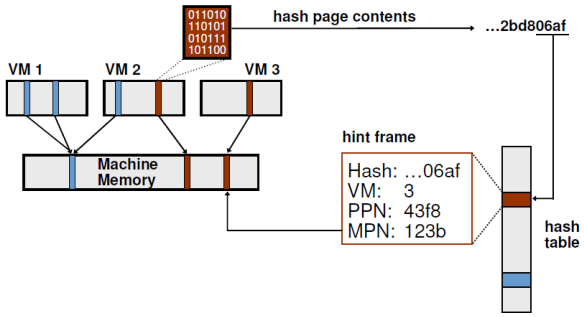
Idle Memory Tax: 75%



- Experiment
  - 2 VMs, 256 MB, same shares
  - VM1: Windows boot+idle
  - VM2: Linux boot+dbench
  - Solid: usage, Dotted: active
- Change tax rate
- After: high tax
  - Redistribute VM1 → VM2
  - VM1 reduced to min size
  - VM2 throughput improves 30%

3.1.10.2.1 Transparent Page Sharing

Page Sharing: Scan Candidate PPN



3.1.10.2.2 Content-Based Page Sharing

Reference:

esx-osdi02-slides.pdf

Memory Resource Management in VMware ESX Server.pdf

3.1.11 HA (High Availability)

• The infrastructure layer At this layer,

VMware HA monitors the health of the virtual machine and will attempt to restart the virtual machine when a failure, such as the loss of a physical host, occurs. This protection is independent of the OS used within the virtual machine.

• The OS layer

Through the use of VMware Tools installed within the OS, VMware HA can monitor the OS for proper operation. This protects against such failures as an unresponsive OS.

• The application layer

With some customization or with a third-party tool, an administrator can also monitor the application running within the OS for proper operation. In the event of a failure of the application, HA can be triggered to restart the virtual machine hosting the application.

Reference:

VMware-vSphere-Evaluation-Guide-1.pdf

3.1.12 VProbe

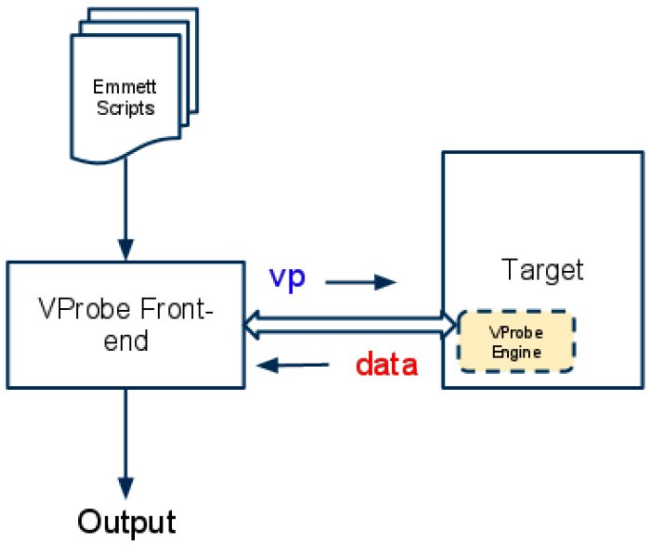


Figure 3.1: Overview of VProbes subsystems

3.1.12.1 ESX VProbes

VProbed VProbed is a daemon process that listens on TCP/IP ports for script using the Emmett
compiler and sends the intermediate VP form to the VProbes Device in the VMkernel for execution. In addition, VProbed also streams output back to the client via the TCP/IP

connection.
VProbes Module A major part of the VProbes backend lives within the Kernel as a Module, holding an output buffer that serializes output data from various VProbes Engines. It has a character device as user interface.
VProbes Engines In an ESX server there is an additional VProbes Engine that is embedded in
the VMkernel. As in the hosted case, the engine is responsible for loading scripts, installing and firing probes, and sending output data to the front end.

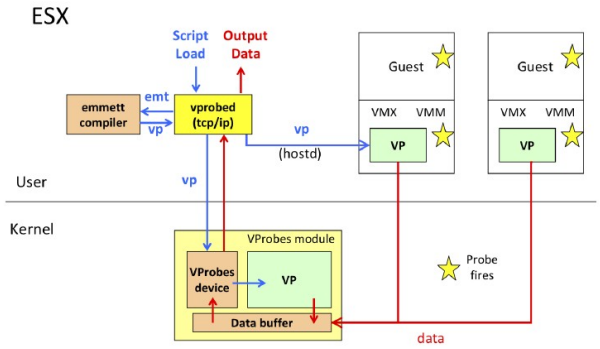


Figure 3.3: VProbes System Architecture, as on ESX

```
string str;

GUEST:system_call {
  syscname(str, RAX);
  printf("%s : %s\n", curprocname(), str);
}
```

Typical cases:

PR574187

Reference:

https://wiki.eng.vmware.com/Vprobe

VProbeGSSTOI.pptx

3.1.13 **VMM VProbes**

3.1.14 **vStrace**

Reference:

vstrace-techtalk.pptx

3.1.15 **Userworlds**

3.1.16 **Debugging**

Performance Counters

- Count low-level hardware events
  - CPU cycles (time)
  - Cache, TLB misses
  - Branch mispredicts
- Many processor-specific differences
  - P3 and P4 can monitor different events
  - P4 has more hardware counters
- Can trigger interrupts on overflow
  - Can configure to count various events
  - Can write counter to control rate

Non-Maskable Interrupts (NMIs)

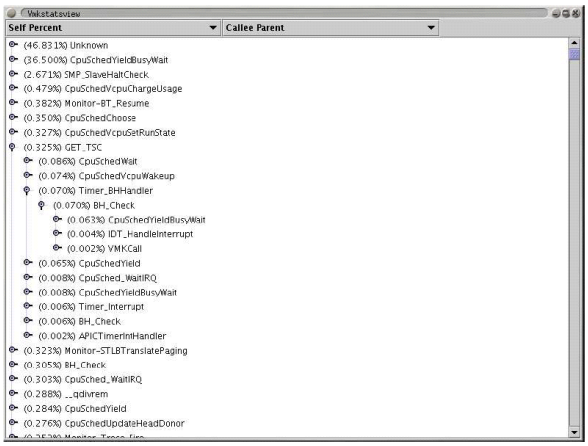
- What are NMIs?
  - Higher priority than normal interrupts
  - Occur even with interrupts disabled
- Performance counters can generate NMIs
  - Eliminates nearly all blind spots
  - Profile code in critical sections

Dangerous Non-Maskable Interrupts

- CPU probably had interrupts disabled for a reason
  - Example: non-atomic manipulation of stack pointer
- Solution: use a Task Gate interrupt handler
  - Isolate NMI processing from profiled code
  - Separate stack, etc.
- Still a few places where we cannot tolerate NMIs
  - Anywhere where the CR3, GDT, and IDT are not all consistent
  - Examples: world switch code, transition to Console OS
- Other than those places, full coverage!
  - VMKernel, Monitor (*), Direct Exec

Implementation Sketch

- Set up performance counters to trigger NMIs
- NMI handler records information for each sample
  - EIP, Stack, PCPU, WorldID, EFlags
- Data collection and storage
  - NMI handler writes raw samples to per-PCPU buffers
  - Bottom-half function drains buffers into global hash table
  - User-level application drains hash table from vmkernel to file



Reference:

vmkstats-rev-2003.pdf

4 Debugging

Reference:

DebuggingESX-Aug-2010.ppt

Esx3x_Logging_Troubleshooting.ppt

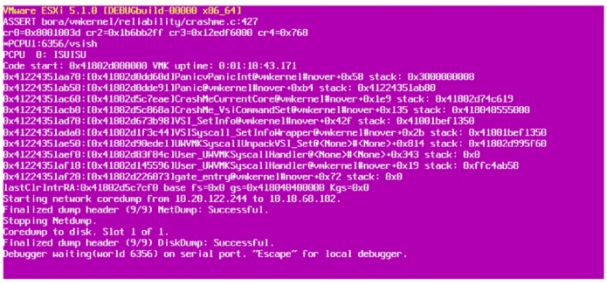
Logs

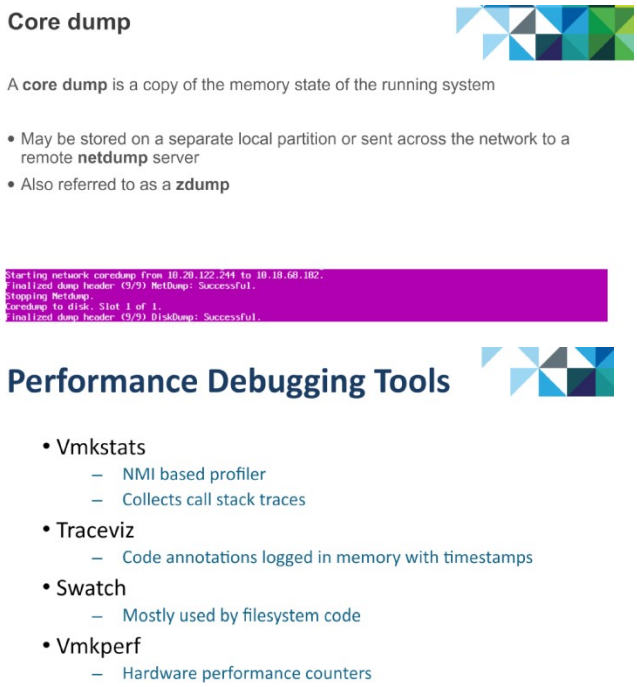
- Always the first place to look!
- All host logs found in directory **/var/log**
- Host logs managed via **esxcli system syslog**

PSOD

Purple Screen Of Death

- System will "PSOD" when it encounters an unrecoverable error
- The host is stopped and provides an information screen



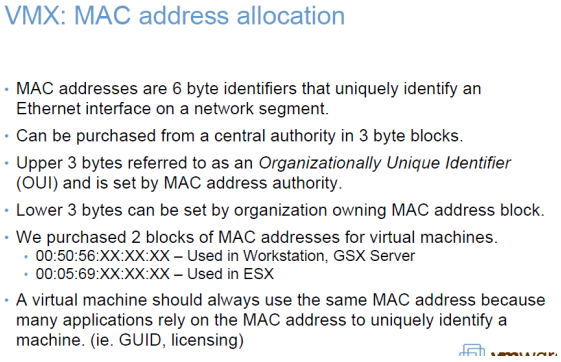


Reference:

Debugging ESXi.pdf

4 Networking

5 Basic



VMX: MAC address allocation

- Divide the MAC address space for 00:50:56:XX:XX:XX into 4 spaces by reserving the upper 2 bits of our MAC address.
- Random (11)
  - Randomly choose a MAC address for remaining 22 bits.
  - On a conflict, randomly choose another one (or just add 1).
- Old IP Based (10)
  - Take a 6 bit hash of the config file path. (CF)
  - Use lower 2 bytes of host's IP address. (A2, A3)
  - Lower 3 bytes -> 0x800000 | (CF << 16) | (A2 | 8) | A3
- IP Based (01)
  - Same as Old IP Based except that 6 bit hash is in the lower order bits.
  - Done because some Ethernet switches hash only the lower 2 bytes of an Ethernet address when routing packets.
  - Lower 3 bytes -> 0x400000 | (A2 << 14) | (A3 << 6) | CF

VMX: MAC address allocation (cont.)

- Fixed (00)
  - User sets MAC address in config file. (ie. ethernet0.address)
  - Fail is user does not set upper 2 bits to 0.
- ESX uses OUI 00:50:69:XX:XX:XX to implement a host based IP scheme that uses an 8 bit hash.
  - Use lower 2 bytes of host's IP address. (A2, A3)
  - Take a 1 byte hash of the config file path. (CF)
  - Lower 3 bytes -> (A2 << 16) | (A3 << 8) | CF
- Conflicts
  - In hosted architecture, checked by the vmnet driver.
  - ESX checks for conflicts by comparing against the MAC addresses of registered VMs that are running or checkpointed. This check is done by having the VMX contact serverd.
  - In all cases, conflicts are resolved by just adding one and trying again.

Reference:

vmware_networking_1001.pdf

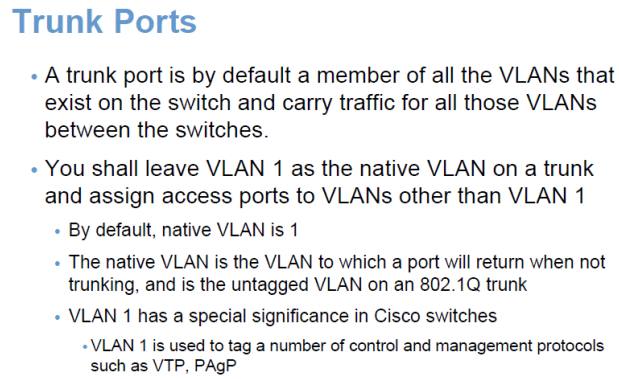
1 VLAN

Types of VLAN

- Port-based VLAN
  - Pros: no switch configuration needed for each workstation.
  - Cons:
    - one VLAN ID per port
    - change configuration when the end-user moves from one port to the other
- MAC-based VLAN
  - Cons: Initial switch manual configuration; performance penalty
  - Pros: automatic tracking is possible thereafter
- Protocol-based
  - e.g., IP, IPX, NetBIOS
  - Layer 3 or 4
  - Flexibility at the penalty of performance and interoperability
- Policy-based VLAN

VLAN Switch Functions

- Learns which VID belongs to which port and updates filtering database
- Bridge database indicates if tagged/untagged frames may be transmitted or received on particular ports
- Adds and removes tags (and recomputes FCS)
- Forwards frames based on VID and 802.1p priority rules



Reference:

vlan-11-05-03_2003.pdf

6 IOV

Chipset Technology MSI-X

Spread the packet processing to multiple cores

Multiple receive queues

Configure queue based on MAC addresses

Assign a receive queue to each virtual NIC

Map receive buffers to guest memory - avoids a copy

Interrupt per queue (MSI/MSI-X) – steer to idle or optimal processor core

Reference:

2007-08-10-howie-xu-slides-000.ppt

7 DVS

Reference:

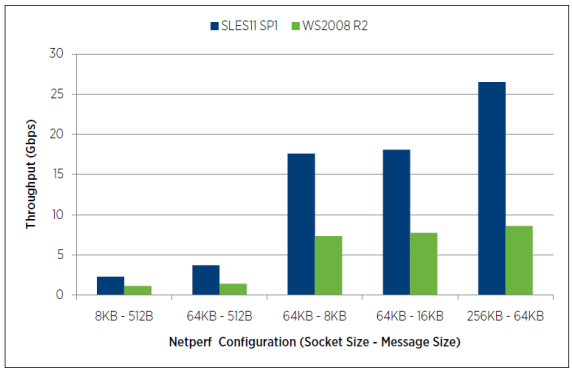
2007-04-20-andrew-lambeth-slides-000.ppt

2008-09-12-andrew-lambeth-slides-000.ppt

8 Performance

VM-to-VM Performance

As the number of cores per socket increases in today's machines, the number of VMs that can be deployed on a host also goes up. In cloud environments with massively multi-core systems, VM-to-VM communication will become increasingly common. Unlike VM-to-native machine communication, VM-to-VM throughput is not limited by the speed of network cards. It solely depends on the CPU architecture and clock speeds.

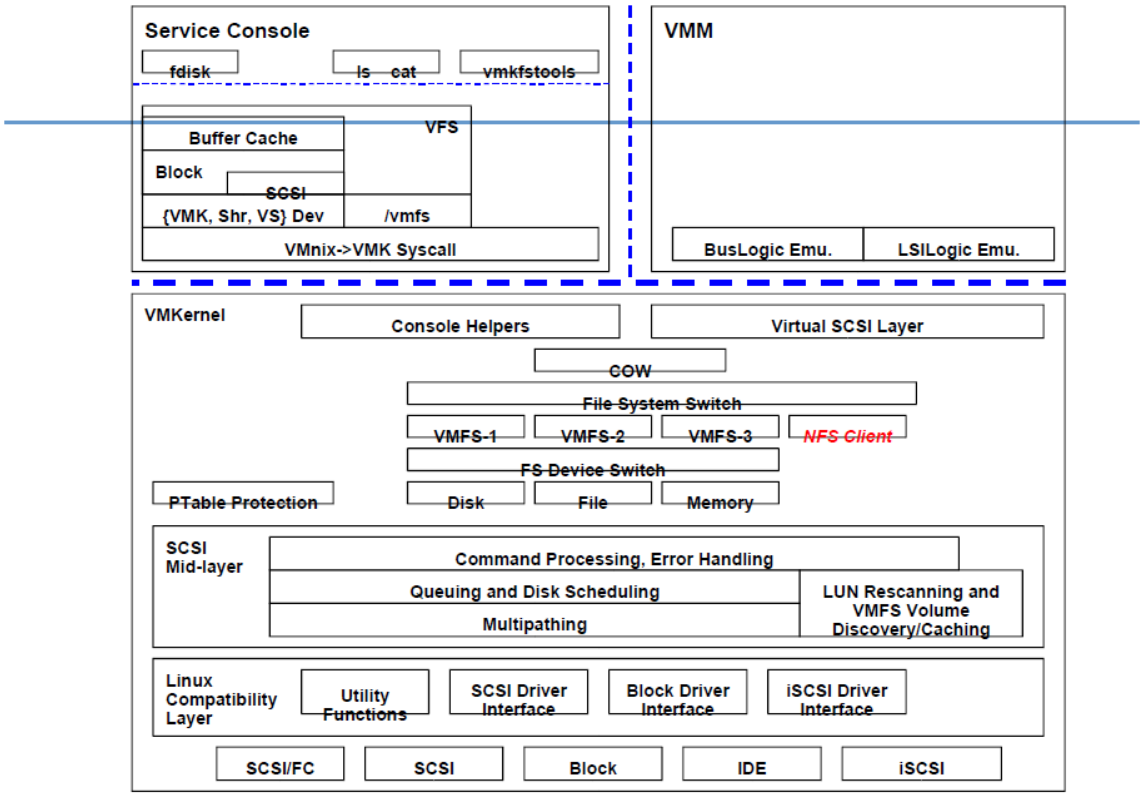


Reference:

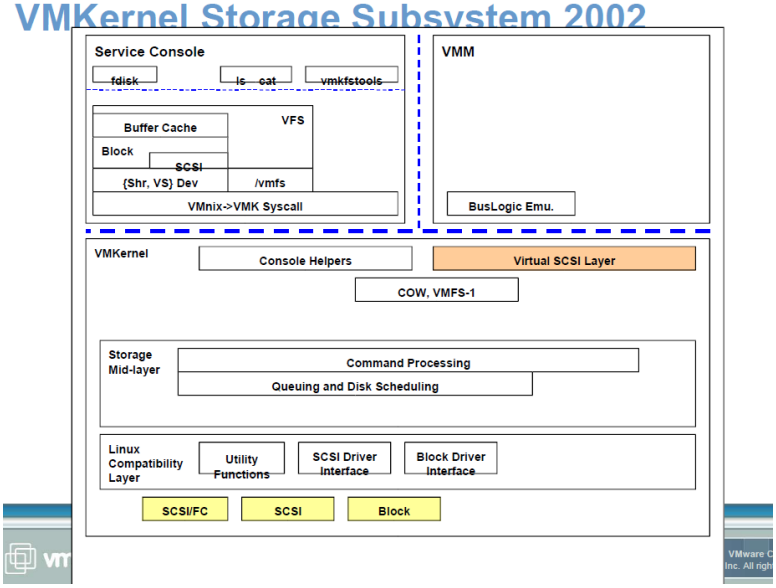
Performance-Networking-vSphere4-1-WP.pdf

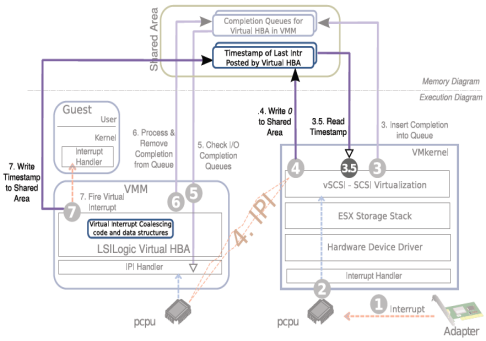
5 Storage

9 Oveview



VMKernel Storage Subsystem 2002





\

Outstanding IOs

Outstanding IO commands - When an application has requested for certain IO to be performed (reads or writes),we've already seen how its broken down into IO commands.These commands are sent to storage devices, and until they return and complete, they are termed as outstanding IO commands.

- Time spent in user/kernel time when accessing FC device is higher (20~30 usec) than that of accessing local RAID device. The difference came from additional overhead of scheduling I/O completion context (kernel threads, VCPUs and so on). For local RAID disk cases, disk I/O is simply accessing PCI cache memory (cached I/Os). So calling and completion context doesn't incur much scheduling related overhead (additional locking and cache misses). This difference is shown in both Native and ESX cases.
- Device time in local RAID disk is estimated by using sampling techniques (not instrumentation). So it could be lower.

Reference:

PerfTalk-Dec13-2012-v1.pptx

https://wiki.eng.vmware.com/NativeFastPathSSD/ESXStorageStackLatencyProfiling

10 DISKIO

Workstation/GSX Server Disk Types (Backends)

- Raw disk
  - An existing partition. Useful for dual booting
- Plain disks
  - One or more fixed sized files residing on host file system. Data stored in files as it would be on a real disk (not in "inscrutable binary format X")
- COW disks
  - Hierarchical, resizable files residing on host file system. Allows us to implement different disk modes

Workstation/GSX Server Disk Modes

- Persistent
  - Behaves as one would expect a real disk to. A write from the guest OS goes directly to the disk/file.
- Undoable
  - Any changes (writes) to the disk are kept in a separate COW disk. At the end of session can be merged with original disk (committed), discarded, or kept around for later (appended).
- Non-persistent
  - Just like undoable disks*, where the user always chooses discard.
    - ** Implementation differences include locking, redo log location and a few other small details*

Raw Disks in More Detail

- Safe raw disks vs. raw disks
  - Raw disks access a single host disk or partition directly
  - Safe raw disks have a .raw file and can access multiple partitions on a disk with varying permissions and other tricks
  - Safe raw disks are what most people mean when they say "raw disks", and are what are referred to in the rest of this presentation.
- The .raw file
  - Contains list of sector ranges and an associated "permission" which can be Read-Write, Read-Only, or No-Access. Allows us to protect partitions not intended for the guest, from the guest.
  - Human readable

Subsystem Architecture - Disklib

- Common interface to all disk backend types
- Understands intermediate commands
  - Read from sector x, write to sector y, read vector etc.
- Decouples our disk implementation from VM implementation
  - Can write stand alone tools for disks

Subsystem Architecture - Disklib

- Common interface to all disk backend types
- Understands intermediate commands
  - Read from sector x, write to sector y, read vector etc.
- Decouples our disk implementation from VM implementation
  - Can write stand alone tools for disks

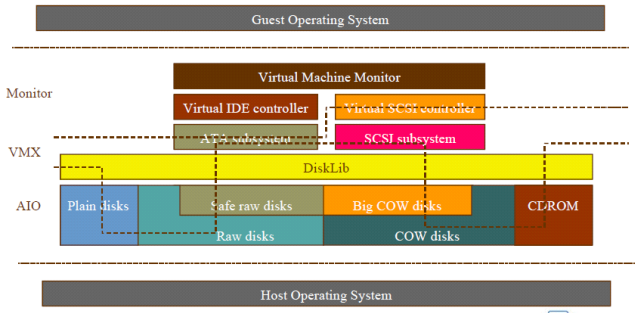
IDE example

- Guest wants to write 2 sectors of data at CHS 1/2/3
  - Writes '1' to IDE Cylinder register
  - Writes '2' to IDE Head register
  - Writes '3' to IDE Sector Number register
  - Writes '2' to IDE Sector Count register
  - Writes memory address to IDE Data register
  - Writes 'CMD_WRITE' to IDE Command register
- IDE emulation translates to:
  - Disklib_Write(sector 222, 2 sectors, dataAddr)
- Backend translates to:
  - lseek("mycow.dsk", 113664)
  - write("mycow.dsk", dataAddr, 1024)
- IDE emulation tells VMM to raise a virtual interrupt

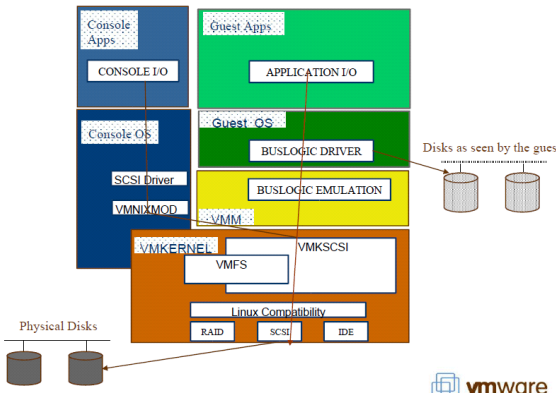
Optimization

- Observations
  - Multiple I/O space accesses to process single disk I/O
  - Too many world switches
  - Many I/O accesses don't have side effects
- Optimization
  - Split emulation between userland and monitor
  - Emulate enough in monitor to only have to go to userland once pe disk I/O in common case
  - (See Jeremy, Ganesh and Beng's paper for detailed analysis of same technique for NICs)

Boxes and Colors



ESX SCSI



Types of Disks

- Entire Disk
  - Pass through commands with S/G mods
  - Non-disk devices (CD, Tape,etc.) are treated similarly
- Partition of a Disk
- VMFS file
- Physical Address for a disk
  - <Adaptername>.<TargetID>.<LUN>.<Partition>
  - <VMFSname>.<Filename>

Controller Emulation

- Emulation of BusLogic controller entirely in VMM
  - Init and SCSI BIOS interaction are done in VMX
- Changes from hosted version
  - All S/G addresses translated to machine addresses
  - I/O pages are pinned to protect against misbehaving VMs
  - No AIO threads, VMKernel-based disk reads are non-blocking
  - VMKernel function calls to issue I/O commands and retrieve results
  - Completion notification through VMKernel to VMM action

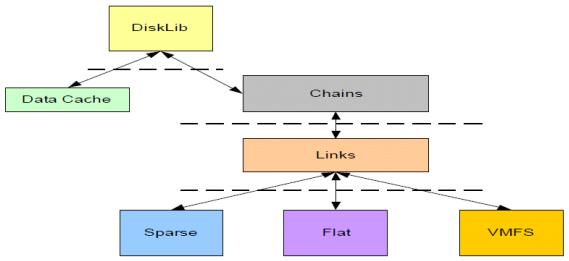
File System - VMFS

- Simple file system designed for small number of large files
- Allow all FS metadata to be kept in memory
  - Large block sizes (>= 1 Mbyte)
  - 1 Mbyte block size => 1 Gigabyte file with one page of metadata
- No caching, all I/O is straight to disk
- Low overhead => high performance
  - Metadata writes only required when file changes in size
  - Metadata reads not required after file is opened

Disk Request Call Path

- VMM:
  BusLogicDATA (Read the data from an OUT)
  *BusLogicCmd_*ScanMailboxes (Read the CCB from the mailbox)
  BusLogicSetupXfer (Set up the Scatter/Gather addresses, pin pages)
  VMK_SCSICommand (Call the VMKernel to issue the command)
- VMKernel
  SCSI_ExecuteCommand (Proceeed based on type of disk, split command)
  VMFS code
  SCSIIssueCommand (Issue command or queue it)
- Linux Compatibility
  SCSILinuxQueueCommand (Attach unique serial number, check for space in adaptor queue)
- Driver
  aic7xxx_queue (Issue to physical device)

DiskLib Organization



- Check Data Cache.
- Call into the chain layer.

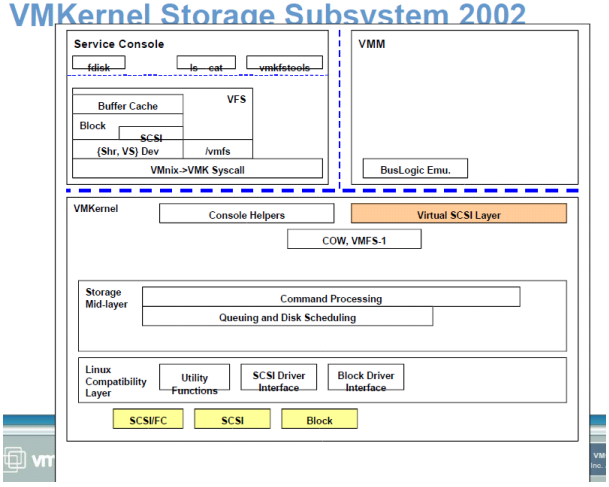
Reference:

diskio_2001.pdf

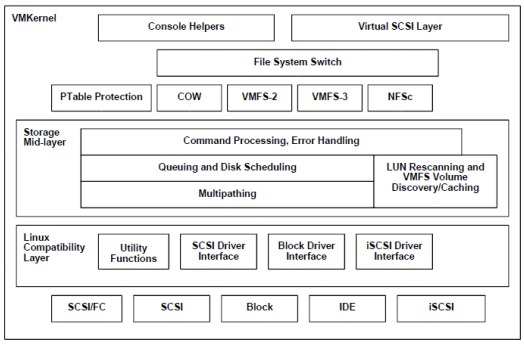
diskperf-2004.pdf

11 VMFS

VMKernel Storage Subsystem 2002



VMKernel Storage Components



VMKernel Storage Components - 1

- Virtual SCSI layer
  - Accepts SCSI requests from VMMs, block requests from COS VBD
  - Forwards requests to FSS (VMFS virtual disks, raw disk mappings) and to the storage mid layer (raw disks, passthru SCSI devices)
- File System Switch (FSS)
  - Accepts requests from virt. SCSI layer, swapper, COS helpers
  - Forwards requests to FS specific implementation (persistent disks, COS files, swap), copy-on-write module (undoable, nonpersistent VMs), VMK storage mid-layer (raw disk mappings, clustering)
- VMKernel storage mid-layer
  - Manages physical adapters on the server
  - Accepts requests from everything on top
  - Command splitting, PAE copies, retries, abort handling

VMKernel Storage Components - 2

- Queuing and disk scheduling
  - Applies scheduling policy to commands from storage mid-layer
  - Manages outstanding commands in per-world, per-target queues
- Multipathing
  - Identifies and abstracts out multiple paths (through HBAs, switches and storage processors) to physical LUNs on disk arrays
  - Does transparent path failover when necessary
- LUN rescanning and VMFS volume discovery
  - Dynamically add/remove LUNs on ESX
  - Up-calls into the FSS to discover/remove VMFS volumes
- Linux Compatibility Layer
  - Rewriting device drivers is boring, porting Linux drivers is easy
- Storage Device Drivers

Why do VMFS?

- Large block size, distributed FS optimized for storing and accessing large files.
  - Keeps VM virtual disk performance close to native SCSI
- Simple format on-disk, simple algorithms
  - For eg, fixed number of files on the volume, block allocator is linear search, only deal with single indirect addressing, no journaling, only handle sector-aligned IO
- Enhanced functionality on SANs
  - Rescan logic, auto-discovery of volumes, hide SAN errors
- Swap and VM consistency
  - VMFS partition protection, exclusive locks across machines, no caching on VM path
- Special primitives for clustered VMs, raw LUNs

VMFS limits

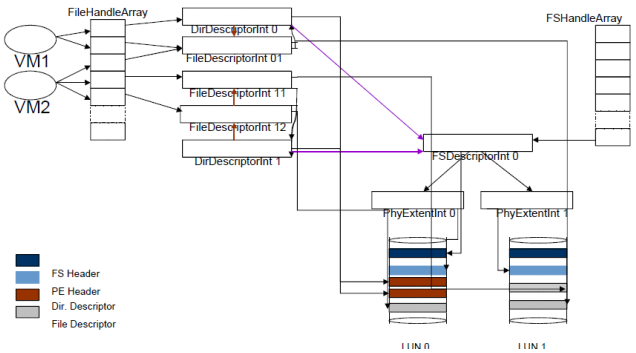
- Max VMFS volumes per ESX server: 128
- Max #Physical Extents (PEs) per volume:
  - VMFS-2 volumes: 32
  - VMFS-1 volumes: 1
- Max PE size: 2TB
- Block size: 1MB to 256MB, in powers of 2
- Max file size:
  - VMFS-2: 456GB to ~64TB, in $2^{(b-1)}$ X 456GB, b=blocksize in MB
  - VMFS-1: 144GB to ~2TB, in $2^{(b-1)}$ X 144GB, b=blocksize in MB
- Default #files: 256 + (n-1) X 64, n = #PEs

VMFS-2 volume format

1. Volume header with FS lock
2. Physical extent (PE) header
3. Fixed number of file descriptors
4. Pointer block bitmap
5. Fixed number of pointer blocks
   (clusters of pointers to data blocks)
6. Data block bitmap
7. Fixed number of data blocks
8. FS-2 recovery region
   (same size as 1 through 6 combined)

Files use single indirect addressing through ptr. blocks

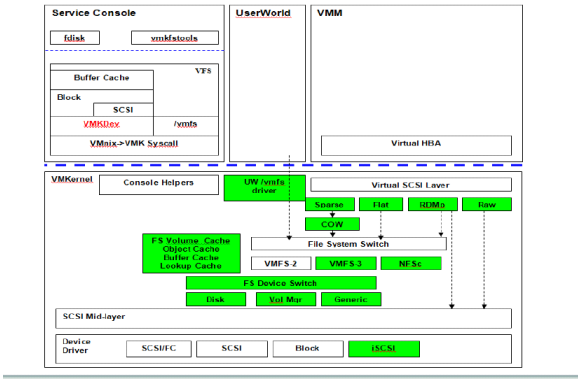
VMFS Data Structures



[2.00+] Raw Disk Mappings (RDM)

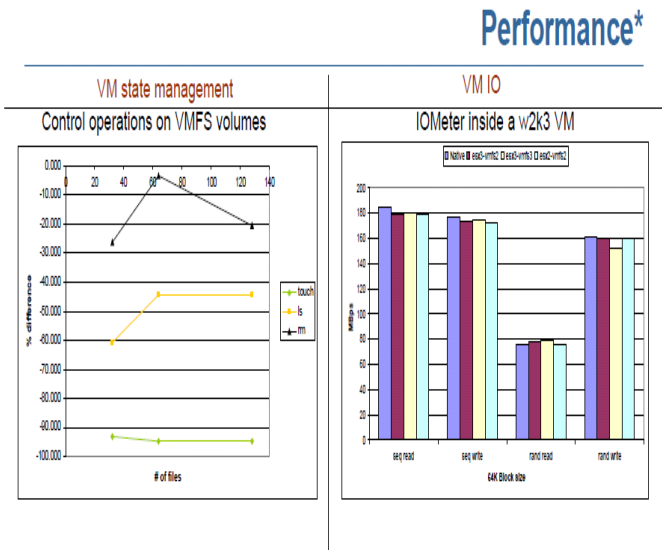
- VMFS file that is a "symlink" to a raw LUN/disk
  - VMFS locking and manageability for raw LUNs
  - Array-based features like snap, replicate, etc. can be used directly on the raw LUN
- Dynamic path and size resolution through LUN IDs
- Another alternative for clustering against physical m/cs
- Undoable, nonpersistent raw LUNs possible through RDMs.
- VirtualCenter will manage/migrate VMs connected to RDMs, but not with (old style) raw LUNs
- RDMs provide SCSI passthru support for SAN mgmt s/w, clustering with physical m/c, etc

ESX3 Storage Stack



New in VMFS-3

- Directories, symlinks
- Sparse files
- Distributed journaling
- Resource clusters
- New lock-lease (heartbeat) mechanism



Reference:

vmfs-2004.pdf

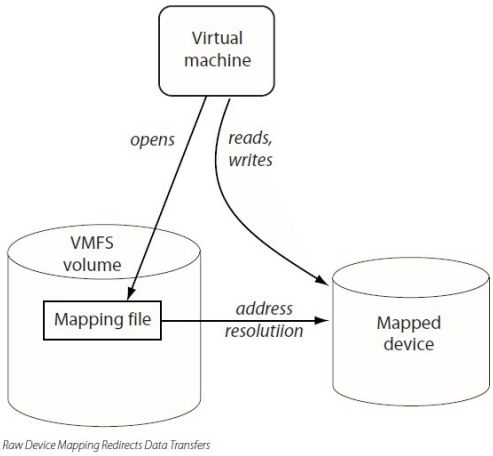
VMFS Update-2005.pdf

VMwareVirtualDiskLayout.pdf

TA09_PVSCSI.pdf

12 RDM

- Raw Disk — A disk volume accessed by a virtual machine as an alternative to a virtual disk
- file; it may or may not be accessed via a mapping file.
- • Raw Device — Any SCSI device accessed via a mapping file. For ESX Server 2.5, only disk
- devices are supported.
- • Raw LUN — A logical disk volume located in a SAN.
- • LUN — Acronym for a logical unit number.
- • Mapping File — A VMFS file containing metadata used to map and manage a raw device.
- • Mapping — An abbreviated term for a raw device mapping.
- • Mapped Device — A raw device managed by a mapping file.
- • Metadata File — A mapping file.
- • Compatibility Mode — The virtualization type used for SCSI device access (physical or
- virtual).
- • SAN — Acronym for a storage area network.
- • VMFS — A high-performance file system used by VMware ESX Server.



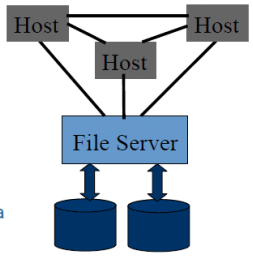
Reference:

esx25_rawdevicemapping.pdf

13 SAN

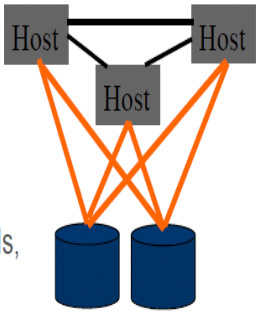
Shared Storage

- Any storage accessible by multiple hosts
- Useful for flexibility, manageability, & high availability
- Types of shared storage:
  - NAS (network attached storage) – files served by a NAS appliance (NetApp) or a file server host
  - SAN (storage area network) – block data accessed directly from disk via Fibre Channel or iSCSI network



SAN (Storage Area Network)

- High-bandwidth network connecting multiple servers to multiple disks
- Provides great flexibility in using raw storage
- Provides greater availability via multiple connections, multiple servers, mirroring
- We will focus first on Fibre Channel SANs, talk about iSCSI SANs later

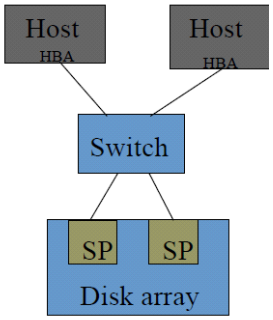


Fibre Channel Networks

- Popular for implementing SANs
- High-speed network (1-4 Gb/s) using either copper or optical, allows distance of 30m to 10km
- Allows hot adding/removal of hosts or storage devices to network
- Low-level protocol that carries SCSI requests and responses
- Fibre Channel drivers typically present the SAN as a SCSI network to each host

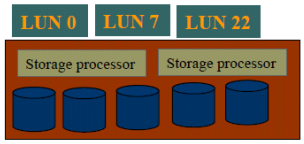
Elements of a SAN

- Servers
- Fibre Channel HBAs (host-bus adapters)
- Fiber optic cables
- Switches – Brocade, McData
- SCSI targets (devices)
  - Tape device, simple disk, disk array
- Disk arrays
  - RAIDed storage carved up into logical disks
  - Multiple storage processors providing RAID



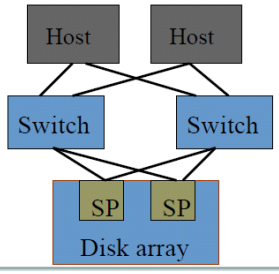
Disk Arrays

- RAIDed storage
  - Large & expandable array of disks
  - Best arrays can deliver 200 Mbytes of disk bandwidth
  - Can be $1,000,000 items
- Can be carved up into many logical disks called LUNs (logical unit number)
- Front-end storage processors implement RAID and LUNs
- Complete redundancy:
  - Redundant storage
  - Redundant power supplies
  - Redundant storage processors



Multipathing

- Multiple paths from host to storage – multiple HBAs, switches, and SPs
- Multipathing module does failover to another path if HBA/SP/switch fails or cable is pulled



ESX View of SAN

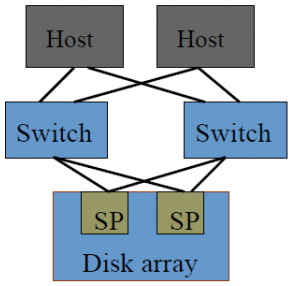
- Fibre Channel disk arrays appear as SCSI targets (devices) which may have one or more LUNs
- On boot, ESX scans for all LUNs by sending inquiry command to each possible target/LUN number
- Rescan command causes ESX to scan again, looking for added or removed targets/LUNs
- ESX can send normal SCSI commands to any LUN, just like a local disk

ESX View of SAN (cont.)

- Must arbitrate to ensure multiple hosts access same disk on SAN safely
  - VMFS-2 and VMFS-3 are distributed file systems, do appropriate on-disk locking to allow many ESX servers to access same VMFS
- SANs/disk arrays are another common resource that must be managed to guarantee certain performance for VMs
  - Add disk bandwidth component to DRS??

Multipathing in ESX

- ESX has built-in multipathing functionality
  - Automatically discovers different paths to the same LUN
  - Causes failover when the FC driver indicates that existing path is dead
- Various policies on choosing the path to storage (fixed, more recently used, round-robin)
- This functionality is provided transparently to the VM, no need for multipathing software in the guest

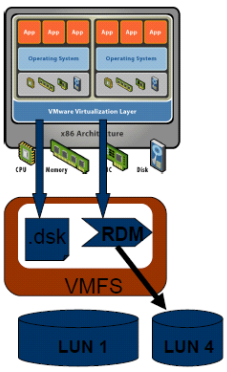


View From the Guest OS

- The guest continues to see a simple virtual SCSI adapter and disk
  - Ensures portability
- Guest does not see any SAN events, but can see some unusual behavior
  - Long pauses and/or dropped commands when failover is occurring or when driver gets reconfiguration event from FC switch
  - Guest OSes may start aborting/resetting because of delayed cmds
  - All I/Os failing if all paths to storage are dead ("all-paths-down")
  - Guest OSes are not expecting their boot disks to have this behavior
- In Dali, user can now hot-add a disk to a VM – fits well with hot-add functionality of SANs

Accessing Raw Disks/LUNs

- Accessing raw LUN is useful if:
  - Some data is already on a LUN, don't want to import to virtual disk
  - Will be sharing data with a physical machine
  - Want to invoke special operations on LUN (e.g. snapshotting)
- Raw disk mapping (RDM)
  - Symbolic link in VMFS to a raw disk
  - Allows a virtual disk to be a raw disk/LUN, rather than a VMFS file
  - Provides naming, permissions, and locking of a VMFS file



Accessing Raw Disks/LUNs (cont.)

- Two kinds of RDMs:
  - Non-passthrough – guest sees vanilla virtual disk, SCSI cmds are still filtered
  - Passthrough – guest sees properties of actual RAW LUN, (almost) all commands are passed through unchanged. Allows invoking special commands like snapshots

SAN Benefits When Using ESX/VC

- General SAN benefits
  - Reliability due to multipathing, RAID, other redundancy
  - Flexibility in all hosts accessing pool of storage
  - Ability to dynamically add and allocate more storage
  - High performance
- SANs provide pool of storage, like VC farms are a pool of CPU power
- With VM disks stored on SAN, any VM can run on any host and easily migrate to any other host

➡ ESX/VC installations have very high use rate of SANs

- SANs also useful for provisioning VMs
  - Allow easy access to libraries of VMs
  - May be able to do the cloning of VM disks in the background
- Disk arrays have long-distance mirroring capability – enable disaster recovery
- SANs are used in blade clusters to provide storage to blades
  - ESX 2.5 and ESX 3.0 support boot-from-SAN

Problems with SANs

- SAN events visible to all hosts
  - Use zoning to partition SAN, contain effects
  - Our SANs are hyper-active because of all the SAN testing going on
- Storage is visible to all hosts
  - Use zoning and LUN masking to expose storage only to appropriate hosts
  - Use VMFS to arbitrate access to shared LUNs, allow any host to access any VM
- Disk array is shared among many hosts, may cause hard-to-analyze performance problems
- LUNs of RAID disk arrays have shared disks underneath

ESX Best Practices

- In general, create VMFSes on fairly large LUNs with extra space, so new VMs can be allocated with talking to SAN administrator
- In Dali, don't pack VMFSes completely full – logs, suspend files, swap files are all on VMFS now
- Disk I/O and VMFS scalability suggest limiting to about 32 active VMs per LUN
  - Less if VMs will use snapshots or will have very frequent power ops or Vmotions
- To allow DRS and flexible VMotion, do zoning and LUN masking so all hosts in a farm see all LUNs

iSCSI Protocol

- iSCSI: protocol that enables SAN connected by Ethernet and TCP/IP rather than Fibre Channel
- Properties of iSCSI
  - 1 Gb/s bandwidth (with contention), 10 Gb/s coming gradually
  - Automatically has security/reliability properties of TCP/IP & SSL
  - Requires more configuration to discover storage devices on net, typically user explicitly names host with iSCSI targets
  - May have more temporary disconnections, because Ethernet network is less robust

iSCSI Support in ESX

- Dali supports:
  - Hardware iSCSI – iSCSI and TCP/IP support in adapter card, so looks like a SCSI card to ESX
  - Software iSCSI – driver in vmkernel supports iSCSI on a basic NIC, uses VMkernel TCP/IP stack
- Unfortunately, iSCSI drivers not too robust, seem like the state that Fibre Channel drivers were three years ago…
- Because of bugs, hardware iSCSI is still experimental
- Software iSCSI uses CPU power on ESX host, so reduces performance
- HW/SW iSCSI can drive up to 60 Mbytes/sec, HW iSCSI should do more when drivers get better

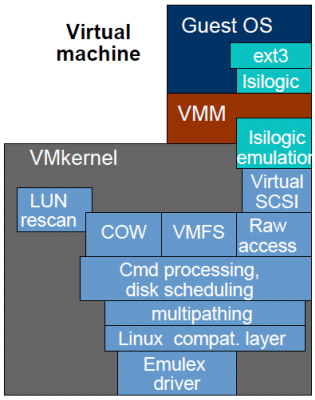
Network-Attached Storage (NAS)

- File server or appliance that serves files via the NFS, CIFS, or Samba protocol
- Client enables access by doing an NFS mount or a Windows file share
- NAS vs. SAN
  - NAS has inherently higher overhead and latency at storage server, since file system code runs in software
  - NAS has CPU overhead at client, running TCP/IP and NAS protocol
  + NAS server manages concurrent access from multiple clients
  + NAS servers (e.g. NetApp) have nice file snapshot model, can simplify backup
  - NAS will be more competitive when TCP/IP overhead on client is reduced via hardware assist

NAS support in ESX

- Only NFS protocol is supported
- We have implemented our own locking protocol (analogous to VMFS) to prevent disk file being opened by multiple hosts
  - Not foolproof, since non-ESX NFS clients don't participate
- Temporary disconnections are more likely – may lead to more aborting & freezes in the guest
- VMotion is supported
- Performance: 30-40 Mbytes/s for single VM, 45Mbytes for several VMs on 1Gb Ethernet

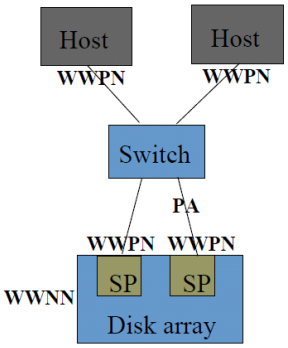
ESX Storage Subsystem



- Guest file system
- Guest storage driver
- I/O emulation in monitor
- Storage stack in VMkernel
  - Virtual SCSI layer
  - File system
  - Cmd processing, error handling
  - Disk scheduling
  - Multipathing
- Linux compatibility layer
- Physical device driver

SAN Naming

- World-wide node name, world-wide port name (64-bit)
  - Hardwired numbers for each possible device on SAN
  - Analogous to MAC addresses for Ethernet cards
- Port address (24-bit) - Analogous to IP address
- FC switches automatically discover WWNs on SAN, assign port addresses



SAN Naming (cont.)

- SCSI target id - used by a host to specify a device
- LUN (logical unit number) – logical disk on a SCSI target
- Fibre channel driver sets up the binding of SCSI target ids to SAN devices (port addresses) when it is loaded
- Unfortunately, this means that the binding of SCSI target id to a SAN device is not permanent
  - Config file can be maintained to provide persistent binding

Reference:

http://engweb.eng.vmware.com/techtalks/VIM-SM-20060421-SAN-iSCSI-NAS.pdf

14 VSAN

Reference:

2011-04-22-vSAN-TechTalk.pptx

15 QOS

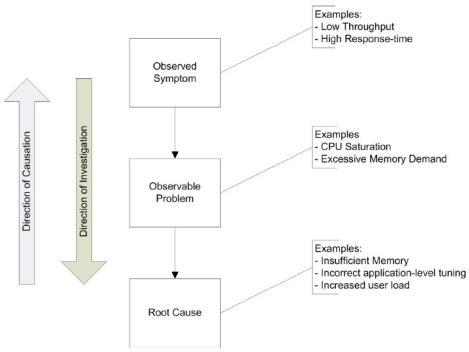
Reference:

storage-resource-pools.pptx

6 Performance

16 Basic

Figure 1. Relationship among Symptoms, Problems, and Cause



1. How do we know when we are done?
2. Where do we start looking for problems?
3. How do we know what to look for to identify a problem?
4. How do we find the root-cause of a problem we have identified?
5. What do we change to fix the root-cause?
6. Where do we look next if no problem is found?

vSphere provides two main tools for observing and collecting performance data. The vSphere Client, when connected directly to an ESX host, can display real-time performance data about the host and VMs. When connected to a vCenter Server, the vSphere Client can also display historical performance data about all of the hosts and VMs managed by that server. For more detailed performance monitoring, esxtop and resxtop can be used, from within the service console or remotely, respectively, to observe and collect additional performance statistics.

Reference:

vsphere41-performance-troubleshooting.pdf

17 VMM

- **Before we move on to the next part of the talk, which is about memory virtualization, let me just make one more quick observation here, related to In/out. As you can see, the hardware assist's higher exit times will have an impact on I/O performance. This is especially true for I/O devices that carry a high rate of traffic in small chunks (i.e., networking).**
- **In fact, choosing a suitable network device is a good way to help alleviate some of these overheads since some devices/driver combinations require that the device be touched fewer times per packet transmitted. For example, a networking device that supports TSO or large packets may be more suitable for deployment in virtual environments. We have also optimized our "paravirtualized" vmxnet device to require as few touches per packet as possible. So if your VM is running with hardware assist, it is better to use vmxnet than e1000.**

Reference:

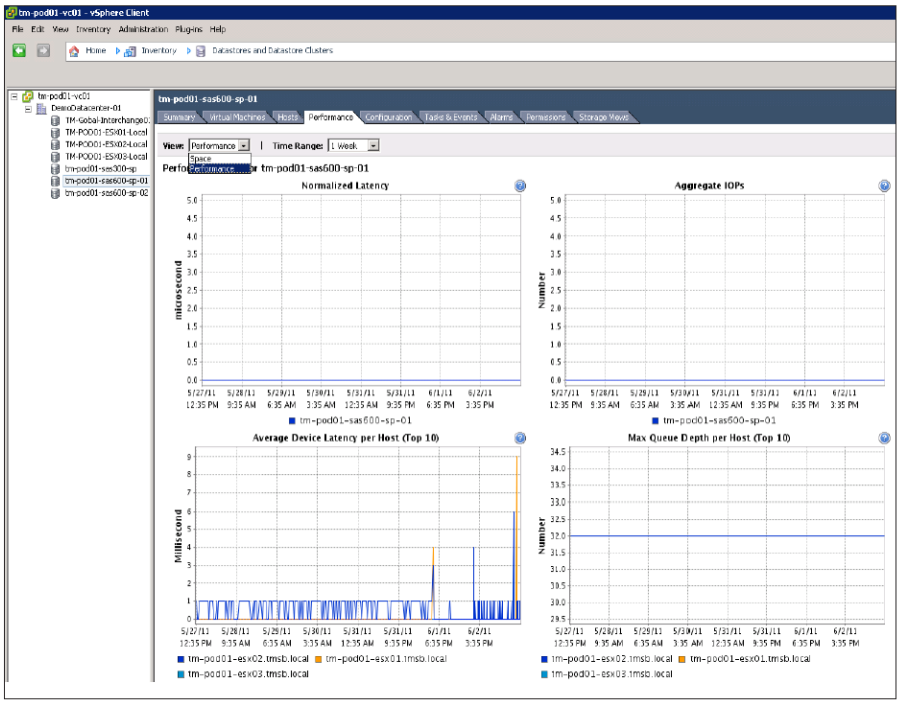
perf-for-gss.ppt

2007-11-09-ole-agesen-slides-000Performance Aspects of x86 Virtualization.ppt

- **Ole's talk on Performance Aspects of x86 Virtualization**
  - http://www.vmworld.com/docs/DOC-2152
  - The actual paper: (I need to find a better link to this!)
  - http://communities.vmware.com/servlet/JiveServlet/download/1147092-17964/PS_TA68_288534_166-1_FIN_v5.pdf
- **Keith and Ole's comparison of HW/SW virtualization techniques**
  - http://www.vmware.com/pdf/asplos235_adams.pdf
- **Performance Group wiki:**
  - http://wiki.eng.vmware.com/Performance
- **Performance Group public blog:**
  - http://blogs.vmware.com/performance/

18 Storage

From viClient



Reference:

VMware vSphere and vCenter resources:

• Product documentation: http://www.vmware.com/support/pubs/

• Online support: http://www.vmware.com/support/

• Support offerings: http://www.vmware.com/support/services

• Education services: http://mylearn1.vmware.com/mgrreg/index.cfm

• Support knowledge base: http://kb.vmware.com

• VMware vSphere® PowerCLI Toolkit Community: http://communities.vmware.com/community/developer/windows_toolkit (or type Get-VIToolkitCommunity within PowerCLI)

• PowerCLI Blogs: http://blogs.vmware.com/vipowershell

Reference:

CPD_PerfTalk_Feb04_2013.pptx

https://wiki.eng.vmware.com/Performance/1M_IOPS

https://wiki.eng.vmware.com/Performance

19 Health Monitoring

CIM

Figure 1. Health Monitoring APIs

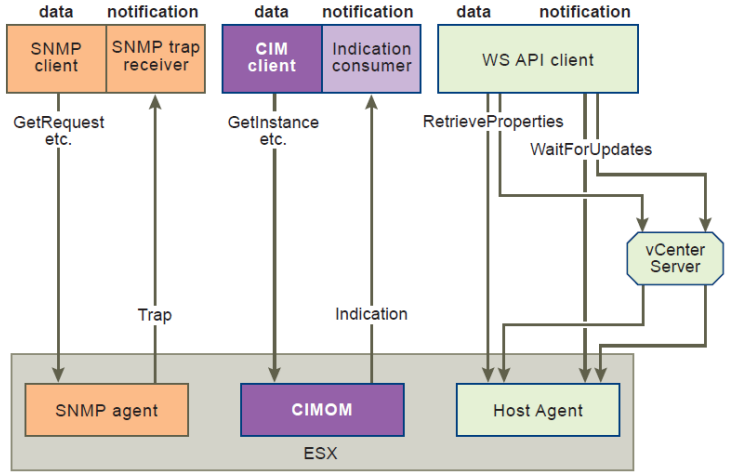


Figure 6. Hardware Health Status Providers on an ESX Host

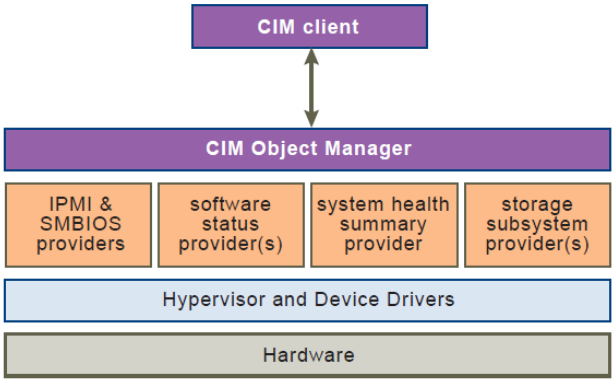
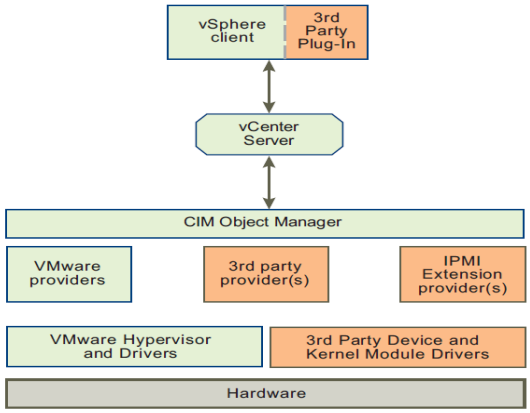


Figure 8. Third-Party Health Monitoring Extensions



20 BenchMark

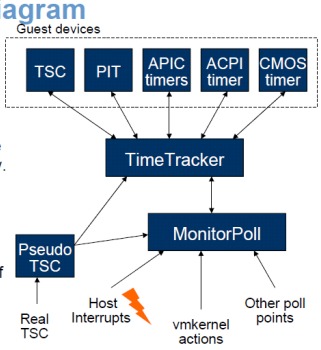
21 Time Keeping

VMware time terminology

- *Real time:* Time as measured by host hardware.
- *Apparent time:* Time as seen within the guest by observing timer devices. Main topic of this talk.
- *Virtual time:* Also known as *VCPU progress*, this is the number of cycles of guest code that have been executed, including cycles spent halting.
  - Somewhat confusing historical term, too deeply ingrained to change. I prefer to say "VCPU progress".
  - Used to keep devices from being too fast or too slow and thus trigger guest races or timeouts.
  - Per-VCPU concept, but some old code is still not SMP-aware and uses VCPU 0's progress as if it were a global clock.

Time tracker block diagram

- TimeTracker interface:
  - Device asks tracker for the apparent time.
  - Device tells tracker apparent time of next interrupt and period, if any.
  - Tracker calls device back when apparent time of interrupt arrives.
- MonitorPoll interface:
  - TimeTracker registers real time of next event.
  - MonitorPoll calls back after real time of event arrives.



Reference:

2009-01-09-garret-smith-slides-000.ppt

TimerVirtualization_2002.pdf

TimerVirtualization2004.pdf

[Timekeeping best practices for Linux guests]

http://kb.vmware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1006427&sliceId=1&docTypeID=DT_KB_1_1&dialogID=890372474&stateId=1%200%20890392590

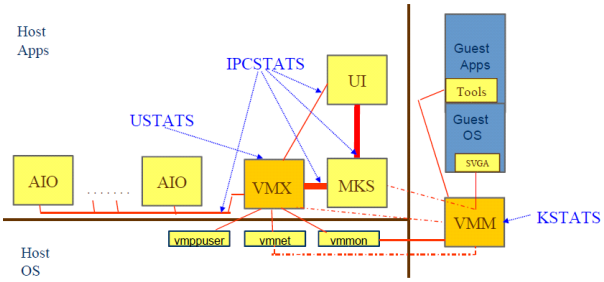
http://www.vmware.com/pdf/vmware_timekeeping.pdf

http://kb.vmware.com/kb/1006427

http://www.vmware.com/pdf/vi3_35/esx_3/r35/vi3_35_25_resource_mgmt.pdf

22 Tools

VMware Workstation



KSTATS: Time in the Monitor (kstats.prl)

```
mhz estimation 551
first=0, last=107, maxid=107

------- KSTATS ------
totSamples = 77183

Category                    %overhd %total       count  avg[us] total[s]
-------------------------------------------------------------------------
TC32_IDENTICAL                49.0   27.1 |           0           209.10
DIRECT_EXEC                   40.2   22.2 |           0           171.66
TC32_RET32_LAUNCH              9.0    5.0 |           0            38.44
TC32_GPCHAIN_JCC0              8.9    4.9 |           0            37.89
None                           5.4    3.0 |    22649786     1.02  23.00
Priv_INOUT                     4.9    2.7 |     2539990     8.16  20.72
MMUAlternateFlush              4.0    2.2 |      892528    19.06  17.01
TC32_REEMIT                    3.8    2.1 |           0           16.07
TC32_MEMIND                    3.0    1.7 |           0           12.75
TC32_RELOCATE_NTOPT            2.8    1.6 |           0           12.14
TC32_VP_SHARED                 2.7    1.5 |           0           11.67
Simulate_VGA                   2.7    1.5 |     5570291     2.06  11.50
Priv_IRET                      2.5    1.4 |     1785518     6.09  10.87
```

USTATS: Time in the VMX

Category	%overhd	%total	count	avg[us]	total[s]
U_Monitor	190.8	53.9%	1517491	537.0	814.88
U_Halt	73.6	20.8%	38723	8118.7	314.38
U_RPC_CursorOn	26.8	7.6%	865115	132.2	114.39
U_RPC_BoundedSyncFIFO	26.3	7.4%	188627	595.8	112.38
U_Disk	6.3	1.8%	36211	738.0	26.72
U_DeviceIRQ	5.2	1.5%	59960	366.9	22.00
U_Poll_IPC	5.1	1.4%	1061273	20.4	21.66
U_Generic	3.8	1.1%	26346	616.5	16.24
U_Poll_Gen	3.0	0.8%	6299845	2.0	12.62
U_RPCTimeQueues	1.8	0.5%	85032	91.7	7.80
U_Poll_LED	1.8	0.5%	14655	519.7	7.62
U_Poll_TimeQueue	1.6	0.4%	196551	34.1	6.71
U_AllocAnon	1.5	0.4%	3068	2086.9	6.40
U_Poll_VIDE	1.1	0.3%	35945	132.9	4.78
U_Poll_Preference	0.8	0.2%	1503	2395.1	3.60
U_ModuleCall	0.7	0.2%	162842	19.5	3.18
U_Timer	0.6	0.2%	77289	34.3	2.65
U_Poll_Keyboard	0.5	0.2%	128009	18.1	2.32

Generated by: `bora/support/scripts/kstats.prl vmware.log`

IPCSTATS: Time Processing IPC Calls

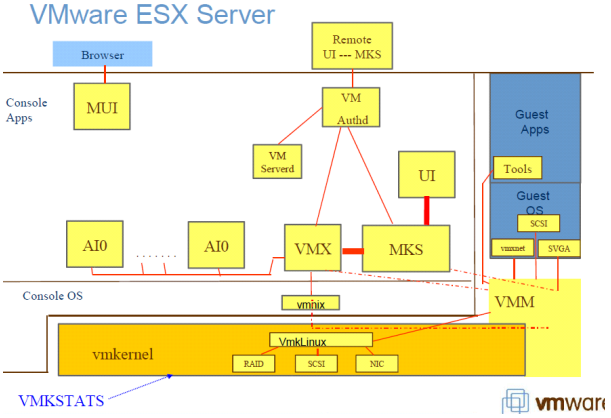
Category	count	Send [us]	Queue [us]	Handle [us]	Reply [us]
SVGAUI_HandleCursorBypass	865115	32.5	3.0	28.5	81.8
SVGA_ReturnValue	188627	92.6	6.0	2.4	0.0
SVGAUI_BoundedUpdateDisplay	188627	32.7	5.1	481.7	0.0
AIOMaster	36586	585.2	12.9	41.2	0.0
AIOSlaveProcessAIO	36586	56.5	7.8	2232.2	0.0
AIOEndPseudoCall	31583	257.0	5.9	2.1	0.0
AIOSlaveStartPseudoCall	31583	82.2	8.8	18.7	0.0
GuiHost_UpdateLED	5716	194.9	24.5	535.1	0.0
MKSVMX_ReturnValue	5430	98.7	8.2	3.4	0.0
MKSBackdoorGetPtr	5406	53.6	6.6	23.7	0.0
GuiHost_UpdateMHz	752	481.7	30.7	1747.6	0.0
Keyboard_InsertMouseEvent	706	978.5	119.0	53.0	0.0
VMXferPrintIPCStats	535	71.4	9.0	832.9	0.0
Usbg_UiStateChange	71	1493.6	3.9	709.6	0.0
MKS_Beep	51	61.8	7.8	961.8	0.0
VMXfer_NOP	34	58.3	7.2	3.7	0.0

Generated by: `bora/support/scripts/kstats.prl -i vmware.log`

How Statistics are Collected

- KSTATS are collected by statistical sampling
  - Code is annotated to mark current KSTATS categories
  - At every timer interrupt, record a sample for the current category
  - Time for each KSTAT: samples * timer-interval
  - Blind spots: code with interrupts disabled
- USTATS are measured with the timestamp counter
  - Code is annotated to mark current USTATS categories
  - Take timestamps when entering and exiting categories
- IPCSTATS are measured with the timestamp counter
  - Record statistics when messages are sent, received and processed
- Statistics are written to the log every 10 seconds
  - Period callback in the monitor calls to user-level

VMware ESX Server



VMKSTATS: Time in the vmkernel (overview)

```
DATA:     /proc/vmware/vmkstats/samples
SAMPLES: 950590
RATE:     2000
OBJS:
    linux.1 = bora/build/obj/server/vmkmod/vmklinux
nfshaper.4 = bora/build/obj/server/vmkmod/SUBDIRS/modules/vmkernel/nf/nfshape
    vmhba.3 =
bora/build/obj/server/vmkmod/SUBDIRS/modules/vmkernel/scsi/BusLogic.o
   vmkernel = bora/build/obj/server/vmkernel/vmkernel
    vmnic.2 = bora/build/obj/server/vmkmod/SUBDIRS/modules/vmkernel/net/3c90x.
-------------------------------------------------------------------------
 SAMPLES    SECONDS  TIME%     CUM% MODULE:FUNCTION
  748572   374.286 78.75%   78.75% vmkernel:Sched_IdleLoop
   96573    48.286 10.16%   88.91% vmkernel:SMP_SlaveHaltCheck
   16122     8.061  1.70%   90.60% linux.1:__delay
   12223     6.112  1.29%   91.89% vmkernel:hash2
    5455     2.728  0.57%   92.46% vmkernel:AllocPageFaultInt
    5344     2.672  0.56%   93.03% vmkernel:CopyFromHost
    4643     2.321  0.49%   93.51% vmkernel:HostSyscall
    3565     1.782  0.38%   93.89% vmkernel:HostEntry
    3541     1.770  0.37%   94.26% vmkernel:HostRetHidden
    3248     1.624  0.34%   94.60% vmkernel:AllocCOWSharePage
    2565     1.282  0.27%   94.87% vmkernel:IOAPIC_MaskVector
```

Generated by: `bora/support/scripts/vmkstats.pl -t bora`

How VMKSTATS are Collected

- VMKSTATS are collected by statistical sampling
  - Set up performance counter to generate interrupts upon overflow
  - Interrupt handler records sample at EIP
  - Uses NMI's so few blind spots
  - Sample data exported via procfs on console OS
  - Finds object files for modules using info exported to procfs
  - Can also annotate code for path data
  - Commands to start/stop, configure events and specify CPUs
- Enable by compiling with SERVER_VMKSTATS=1

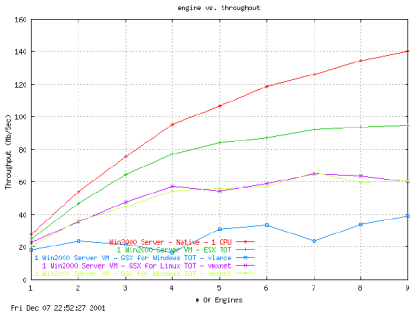
The Picture is Incomplete

- Need to use host tools to see host effects
  - Perfmon on Windows
  - Top, ps, free, … on Linux
- Visibility into user-level processes other than VMX
  - Can currently use gprof or vtune
- Missing the big picture
  - Interaction between processes

Benchmarks, Benchmarks, Benchmarks

- Microbenchmarks
  - CPU: getpid, context switch, segv, …
  - Disk: disk streamer
  - Network: ping, netperf
- Small workstation workloads
  - Various OS boot/halts, Volano, runmake
  - ftp, file copy, Windows search, java compiler, SPEC, Winstone
- Interactive performance
  - Mouse movement, basic Windows tasks, MS Office apps
- Large server-class workloads
  - Squid web cache, Exchange, Netbench, Webbench

Benchmark Results: Netbench



http://vmweb.vmware.com/~rajit/results/netbench

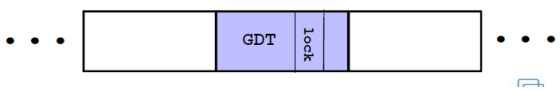
vmware

Identifying Performance Problems

- No fixed methodology
- Is the workload CPU bound?
  - Use kstats.prl and vmview to determine virtualization overheads
  - Is the problem in the monitor, at user-level or in the vmkernel?
  - Look at CPU utilization compared to native
- Is it networking or disk?
  - Break down virtualization overhead by packet or Mbyte read/written
- Is it the host?
  - Rely on host tools

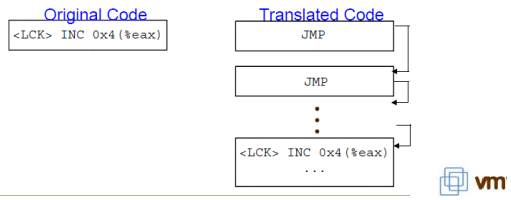
Performance Analysis Case Study

- Problem seems to be in the binary translator
- Turn on logging to see which EIPs are being translated
  - Same few EIPs are being retranslated frequently
- Retranslating due to trace faults:
  - Guest is writing to the same page as the GDT
  - Look at Linux loadmap to find culprit functions
  - Idle thread's `file_lock` variable being written
  - Lock prefix on instructions trigger retranslation



Performance Analysis Case Study

- Why does retranslation cause increase in time spent executing in TC?
  - Forwarding jumps put in from old translation to new translation
  - Results in long jump chains (~3000 jumps!)
  - Ramps in vmview when idling correspond to TC flushes



Reference:

2001performance_analysis_tools.pdf

7 VMotion

23 Requirements for VMotion

In order to facilitate VMotion between hosts, each host must meet certain basic requirements:

 Datastore compatibility: The source and destination hosts must use shared storage. You can implement this shared storage using a SAN or iSCSI. The shared storage may use VMFS or shared NAS. Disks of all virtual machines using VMFS must be available to both source and target hosts.

 Network compatibility: VMotion itself requires a Gigabit Ethernet network. Additionally, virtual machines on source and destination hosts must have access to the same subnets, implying that network labels for each virtual Ethernet adapter should match. You should configure these networks on each ESX host.

 CPU compatibility: The source and destination hosts must have compatible sets of CPUs. This requirement is discussed in detail in this guide.

24 Migration of a virtual machine

The process of moving an entire virtual machine from one host to another.

Complete virtualization of all components of a machine, such as CPU, BIOS, storage disks, networking, and memory allows the entire state of a virtual machine to be captured by a set of data files. Therefore, moving a virtual machine from one host to another is, in essence, a specialized data transfer between two hosts. Based on the state of the virtual machine during migration, the migration is referred to as cold or hot.

 Cold migration: Migration of a virtual machine that has been powered off on the source host. The virtual machine is powered on again on the destination host after the transfer of the virtual machine state is completed.

 Hot migration: Migration of a virtual machine that is powered on. The virtual machine (and applications) previously running on the source host continue execution on the destination host, without detecting any changes, after the hot migration is complete. Based on availability of the virtual machine during migration, hot migration falls into the following subcategories:

 Suspend/resume migration: Involves suspending a running virtual machine, then resuming the suspended virtual machine on a different host.

 Live migration (VMotion): Migration of a running virtual machine from a source host to a

destination host without any disruptions (downtime) to the running virtual machine.

25 Applications of VMotion

The ability to migrate a running virtual machine across different hosts without any perceivable impact to the end user opens up a wide array of applications. Some important applications of VMotion are:

 Hardware maintenance: VMotion allows you to repair or upgrade the underlying hardware without scheduling any downtime or disrupting business operations.

 Optimizing hardware resources: VMotion lets you move virtual machines away from failing or underperforming hosts.

 VMware product upgrades: VMotion, coupled with disk migration capabilities, allow you to upgrade from an older version of VMware ESX. You can also use this feature to upgrade between incompatible versions of the VMware Virtual Machine File System (VMFS) while keeping the virtual machine live.

VMware VMotion and CPU Compatibility

 VMware Distributed Resource Scheduler: VMware Distributed Resource Scheduler (DRS) continuously monitors utilization across resource pools and allocates resources among virtual machines based on current needs and priorities. When virtual machine resources are constrained, DRS makes additional capacity available by migrating live virtual machines to a less-utilized host using VMotion.

26 Compatibility

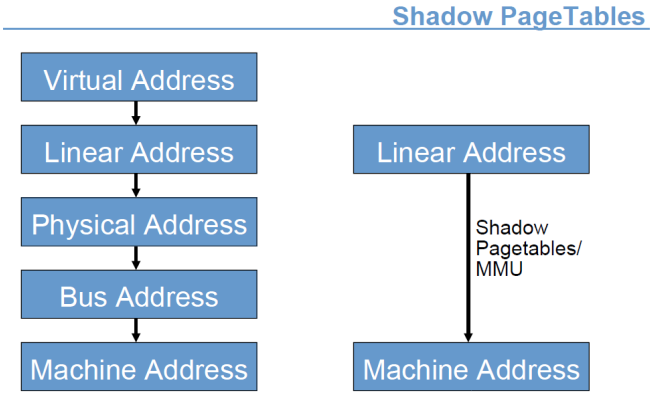
Reference:

vmotion_info_guide.pdf

8 HA & FT

9 Memory

27 BusMem



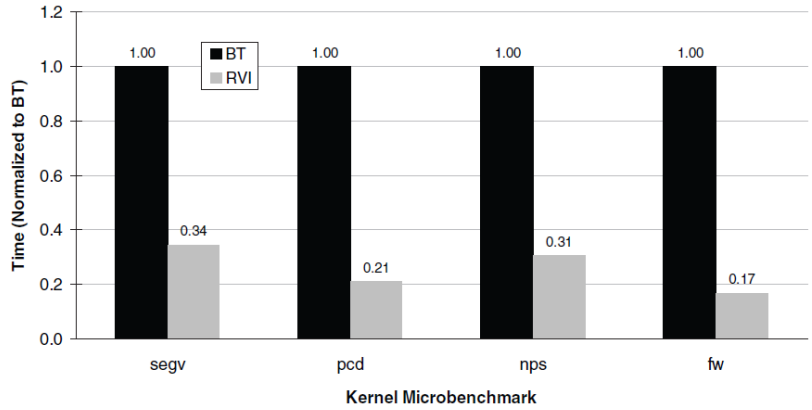
28 RVI

Benchmarks

In this study we used various benchmarks that to varying degrees stress MMU-related components in both
software and hardware. These include:

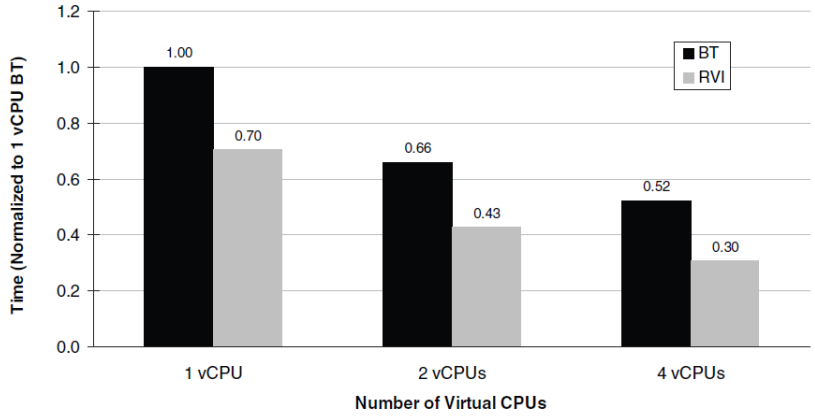
- Kernel Microbenchmarks: A benchmark suite for system software performance analysis.
- Apache Compile: compiling and building an Apache web server.
- SPECjbb®2005: An industry standard server-side Java benchmark.
- SQL Server Database Hammer: A database workload that uses Microsoft SQL Server on the backend.
- Citrix XenApp: A workload that exports client sessions along with configured applications.

Figure 4. MMU-Intensive Kernel Microbenchmark Results (Lower is Better)



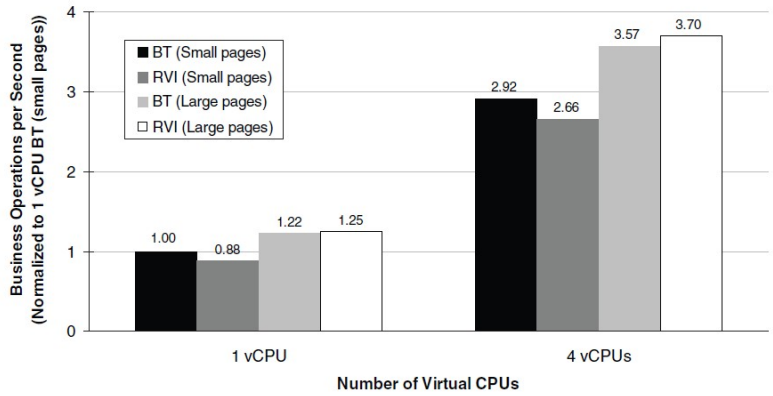
Kernel microbenchmarks comprise a suite of benchmarks that stress different subsystems of the operating
system. These microbenchmarks are not representative of application performance; however they are useful
for amplifying the performance impact of different subsystems so that they can be more easily studied.They
can be broadly divided into system-call intensive benchmarks and MMU-intensive benchmarks.
segv: A C program that measures the processor fault-handling overhead in Linux by repeatedly generating and handling page faults.
 pcd: A C program that measures Linux process creation and destruction time under a pre-defined number
of executing processes on the system.
 nps: A C program that measures the Linux process-switching overhead.
 fw: A C program that measures Linux process-creation overhead by forking processes and waiting for them to complete.

Figure 5. Apache Compile Time (Lower is Better)



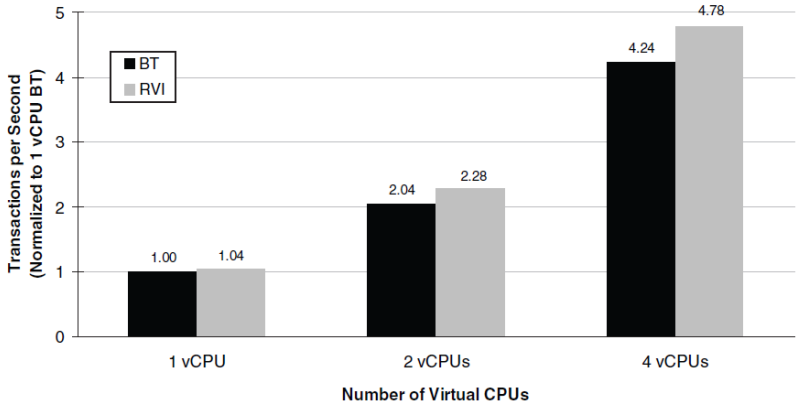
This particular application is at an
extreme of compilation workloads in that it is comprised of many small files. As a result many short-lived
processes are created as each file is compiled. This behavior causes intensive MMU activity, similar to the
MMU-intensive kernel microbenchmarks,

Figure 6. SPECjbb2005 Results (Higher is Better)



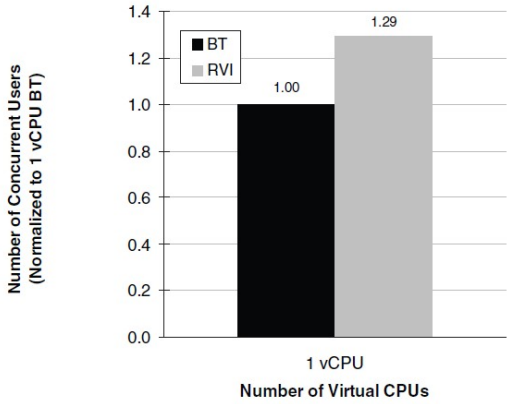
SPECjbb2005 is an industry-standard server-side Java benchmark. It has little MMU activity but exhibits high
TLB miss activity due to Java's usage of the heap and associated garbage collection. Modern x86 operating
system vendors provide large page support to enhance the performance of such TLB-intensive workloads.

Figure 7. SQL Server Database Hammer Results (Higher is Better)



Database Hammer is a database workload for evaluating Microsoft SQL Server database performance.

Figure 8. Citrix XenApp Results (Higher is Better)



Citrix XenApp is a presentation server or application session provider that enables its clients to connect and

run their favorite personal desktop applications. To run Citrix, we used the Citrix Server Test Kit (CSTK) 2.1
workload generator for simulating users. Each user was configured as a normal user running the Microsoft
Word workload from Microsoft Office 2000. This workload requires about 70MB of physical RAM per user.
Due to heavy process creation and inter-process switching, Citrix is an MMU-intensive workload.

AMD RVI-enabled CPUs offload a significant part of the VMM's MMU virtualization responsibilities to the hardware, resulting in higher performance. Results of experiments done on this platform indicate that the current VMware VMM leverages these features quite well, resulting in performance gains of up to 42% for MMU-intensive benchmarks and up to 500% for MMU-intensive microbenchmarks.
We recommend that TLB-intensive workloads make extensive use of large pages to mitigate the higher cost of a TLB miss.

Reference:

RVI_performance.pdf

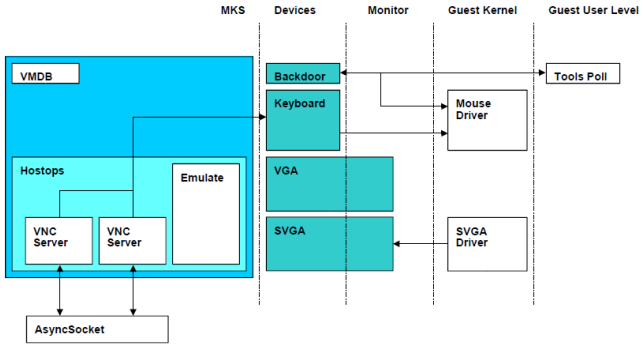
10 VMX

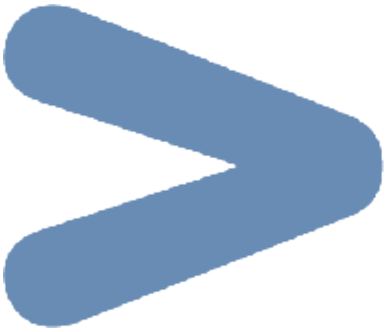
29 Userworlds

- Similar to a traditional “process” or “task”
- Each *thread* is a VMKernel World. Threads are members of a *cartel*.
- No inter-process support
    - No fork, no inter-process signals, no shared memory, no inter-process locks/semaphores.
- No (useful) /proc or /dev
- No Linux services/daemons
- **Will become less and less like Linux**
- Simple Linux apps work just fine
    - Threads, signals, files, mmap, debugging
- VMX loads up, and fails on access to /tmp
    - Makes ~80 syscalls to this point
- POSIX threads via linuxthreads library
    - Built on clone, wait4, signals, and pipes
- POSIX signals/”real-time” signals
- File descriptors
    - Objects: Files, sockets, RPC connections, pipes, directories
    - Ops: read, write, open, close, lookup, stat, setStat, poll, mmap
    - Files on /vmfs/*whatever* opened natively

30 MKS

MKS Architecture – VNC Server





The next step beyond a dumb framebuffer card: 2D accelerators.

- **Started appearing in the early 1990s**
- **No standard hardware interface, hardware specs generally hard to get**
- **Main features:**
  - Adds a co-processor (Three “threads”)
    - CPU writes to VRAM *or* CPU writes to command FIFO
    - GPU executes commands, reading/writing VRAM
    - On-screen VRAM scanned out to VGA/DVI port.
  - Accelerated commands:
    - Fill
    - Blit (bit block transfer)
    - Line drawing
    - etc...
- **This same basic accelerator architecture underlies pretty much all modern GPUs**
- **Memory is now non-uniform**
  - VRAM is for more than just the framebuffer
  - Many places a given object can live

        - System
        - AGP
        - "VRAM"
        - On-chip caches
- **Asynchrony**
    - GPU and CPU are co-processors, sharing resources
    - Asynchronous DMA transfers
    - All resources must be tracked in space and time
    - Methods to synchronize CPU and GPU: fences, IRQs, multiple hardware contexts, etc.
- **Special purpose hardware**
    - Video overlay
    - Scaled blits
    - Motion compensation (MPEG)
    - Colorspace conversion
    - Triangle rendering…
- **Big picture: Lots more tradeoffs**
    - Time/memory, CPU vs GPU vs Dedicated hardware.

- **This is a fairly high-level look at VMware's graphics stack**
    - Not a 3D-specific diagram- same parts are used for:
        - Unaccelerated 2D
        - Accelerated 2D
        - Video
        - 3D
- **Follows the patterns presented earlier**
    - "Virtual Graphics Processor" model
- **Data flow**
    - App requests a drawing operation
    - Guest API queues it
    - Guest kernel module converts to VMware SVGA format
    - Written to Command FIFO (shared memory)
    - MKS (a thread in the VMX) reads commands from the SVGA device's FIFO, carries them out

- Uses “host ops” tree, with platform specific rendering backends.
- Other backends for things like VNC (remote console)

- **Memory:**
  - Command FIFO: Just like on a physical GPU, stores asynchronous accel commands
  - Guest VRAM: Host system memory, mapped to guest as PCI MMIO
  - Host VRAM: Not directly visible to guest (Indirect access via DMA)
- **Asynchronous Threads:**
  - Guest: Writes command FIFO, writes/reads DMA buffers and shadow FB
  - Host: Converts FIFO commands to physical GPU, writes/reads DMA buffers and shadow FB
  - Physical GPU: Carries out commands from the MKS, manipulates host VRAM
  - Display: Refreshing from physical VRAM

- **A quick look at the steps taken to bring 3D to our formerly 2D-only infrastructure**
- **3D is much more demanding than 2D...**
  - Performance/concurrency. With 3D, MKS spends most of its time busy. Scheduling and asynchronous execution is key.
  - Dynamic MKS backends. Spin up 3D support only when we need it.
  - FIFO efficiency. Lots more command traffic now...
  - Memory management. There are more kinds of memory now (host VRAM, guest VRAM, GMR) and we need good ways to access each.
  - Debug tools!
    - 3D is complicated, and it is fairly opaque
    - Hard to deduce root cause by looking at faulty output
    - (Especially for performance problems!)
    - We developed sophisticated whole-system tools for:
      - Logging
      - Profiling
      - Interactive debugging
- **Compositing 2D and 3D**
  - With 2D, we had a single framebuffer, a single type of memory
  - Now we have multiple sources of on-screen data:
    - 2D – Guest VRAM (system memory)
    - 3D – Host VRAM (on graphics card)
    - Mouse cursor
    - Video overlays
  - Very expensive to just render 3D back into system memory

- Need to composite final image on GPU
- No real hardware does it quite like this.. Interaction with guests can be tricky.

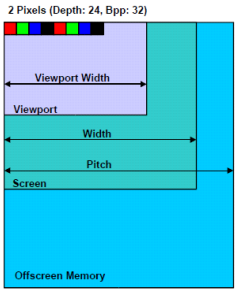
- **SVGA3D Protocol**
  - GPU-neutral, API-neutral
  - Scope:
    - Drawing commands
    - Shaders
    - Memory management
    - DMA transfers
    - Configuring 3D pipeline
  - Inspired by Direct3D, but cleaned up / idealized
    - Thin interface
    - Easy checkpointing and portability
- **DirectDraw / Direct3D drivers for the guest**
  - Conceptually simple, but...
  - Lots of code. Microsoft HAL interface is very complex and full of backward compatibility kludges.
  - Manages DMA buffers, rendering, and synchronization between 2D and 3D subsystems
- **MKS backends for SVGA3D**
  - Implement idealized protocol on top of real APIs
  - OpenGL and Direct3D implementations
    - OpenGL very complicated, but mature – Linux / Mac hosts
    - Direct3D simple and efficient, but still young – Windows hosts
      - Why Direct3D? 3rd party Direct3D drivers are phenomenally better than OpenGL drivers across the board.
  - These backends are the most interesting part of our 3D implementation
    - We'll cover them in detail now...

Mouse/Keyboard Primer

- **PS/2 Port**
  - 2 registers (0x60 for data, 0x64 for status) and IRQ 12
  - USB devices exist, but don't matter.
- **Keyboard**
  - Write scan codes to data port.
- **Mouse**
  - Sitting on the desk next to the keyboard
  - 4, 1-byte writes for dx, dy, dz, and button state.
  - Host transforms raw PS/2 packets and draws cursor.
  - May do strange things (e.g. 5 button mice), but that doesn't matter.

Screen Primer – Properties

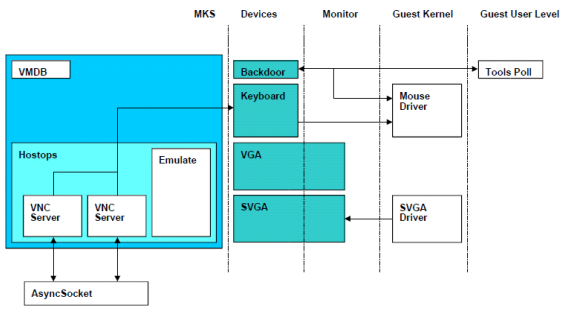
- **Glossary of Terms**
  - VRAM : Video RAM.
  - Screen : The portion of VRAM that's potentially visible.
  - Viewport : The visible portion of the screen.
  - Offscreen Memory : VRAM minus the Screen. Typically used for caching images.
  - Pixel : A single dot on the screen
  - Bits Per Pixel (bpp) : Pixel size in bits of VRAM.
  - Depth : Number of significant color bits per pixel.
  - Pitch : Distance between two vertically aligned pixels.



Screen Primer – Display Types

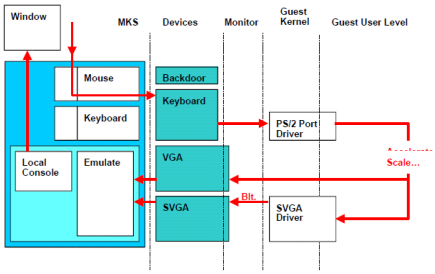
- **VGA**
  - The greatest common denominator
  - Multiple text modes and color modes.
    - 640x480 16 color, 320x240 256 color, etc.
  - Banked : i.e. a single pixel on the screen is described by multiple discontiguous values in video memory.
- **SVGA**
  - "Super VGA". Bigger resolutions and more color depths.
  - Every vendor implementation is different, proprietary, and largely unpublished.
- **VESA BIOS**
  - Post hoc attempt at SVGA standardization.
  - Real mode or 16-bit protected mode interfaces program display hardware.
  - Popular for DOS games
  - Offered as a "last resort" by some OSes where vendor support is lacking.

MKS Architecture – VNC Server



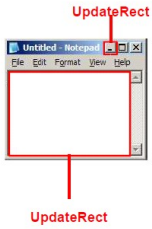
MKS Mouse Internals – Software Cursor

- Mouse Types
  - Software Cursor
- Properties
  - Uses guest mouse acceleration
  - Cursor serialized with other drawing
  - Best you can do for VGA.
  - High latency.
  - Low predictability.



SVGA : Basic Drawing

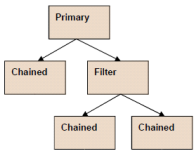
- Drawing Requirements
  - Shadow FB must be 100% accurate (guest read back, host exposure)
  - Host screen must be kept up-to-date.
- Basic Drawing
  - Guest draws everything
  - SVGA Driver notifies host via an UpdateRect.
  - Host periodically wakes up and copies update region to screen.
- Basic Drawing Cost
  - 2 * w * h memory copied.
  - Already an improvement over trace based drawing
- Example



MKS HostOps

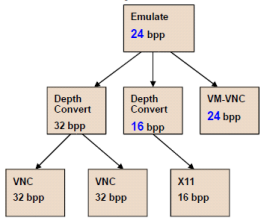
- Must at least…
  - Emulate onto the shadow frambuffer.
- …and may have to.
  - Draw to the host screen.
  - Do color space transformations
  - Send data to a remote console.
- Structure
  - Pure, abstract classes VideoHostOps, ChainedHostOps, etc.
  - Concrete, derived classes EmulateBackend, etc.
  - Primary – Root of a tree. Manages a framebuffer.
  - Chained – Renders a framebuffer.
  - Filter – Both.
- MKSHostOps vtable Interface

```
typedef struct VideoHostOps{
  void (*rectROPFill)(VideoHostOps *self, ...);
  void (*update)(VideoHostOps *self, ...);
} VideoHostOps;
typedef struct ChainedHostOps {
  VideoHostOps parent;
  void (*setFB)(ChainedHostOps *choSelf, FBDesc fbDesc);
} ChainedHostOps;
```



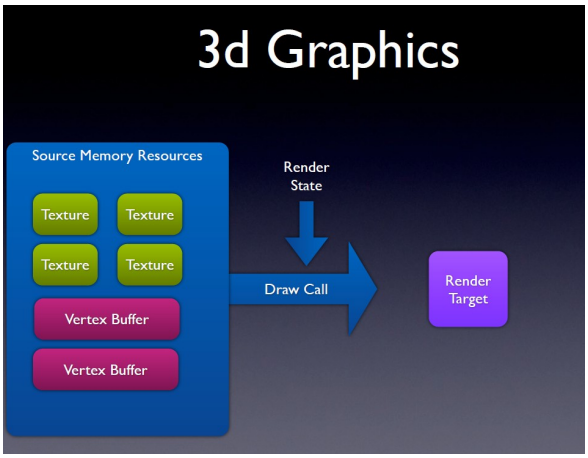
MKS HostOps : Sample #3

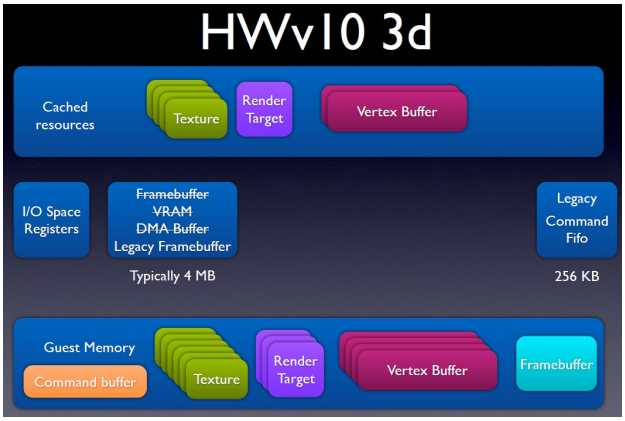
- VNCBackend
  - Renders to the network
  - Forwards upstream mouse and keyboard commands to device layer.
- MKSHostOpsChainManager
  - Manages the tree of nodes
  - Builds optimal structure
  - Owns node iterators
  - Does all sorts of neat tricks
- Bag O' Tricks
  - Builds optimal render trees.
- MKSHostOps Structure



Reference:

mks-2004.pdf





Reference:

3d-presentation.pdf

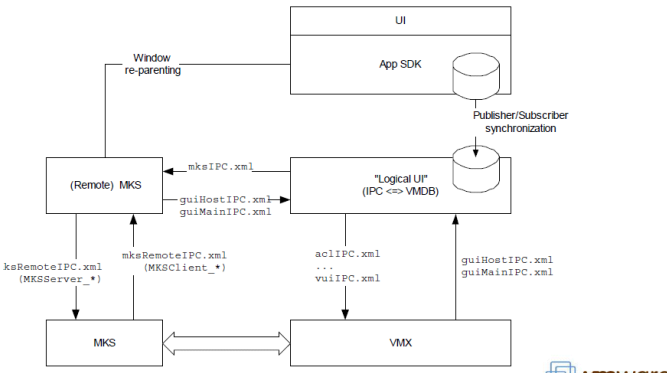
31 VMDB

The VMDB

- A hierarchical namespace, similar to the Windows registry
  - Basic functionality includes Get/Set operations and Set callbacks
  - All names must be specified in a schema
  - Has arrays
- The VMDB context
- Callback semantics
  - All callbacks are asynchronous. So, only Set callbacks
  - Previously queued callbacks can be merged with new callbacks
- Different namespace sub-trees can “live” in different instances of VMDB in different processes
  - Publisher vs. subscriber VMDB instances
- Access control

vmwo

Application restructuring – phase I



Phase I status

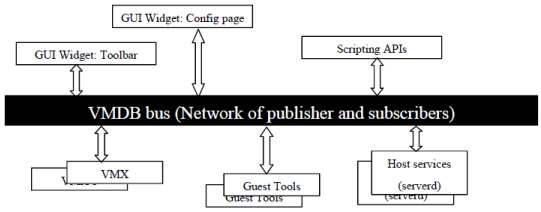
- The VMX functionality is exposed via the VMDB namespace.
- There is a VMDB schema that specifies the VMX functionality, which has been exposed.
- There are COM bindings for all VMDB interfaces. There is also an ActiveX control that encapsulates the MKS.
- It is possible to write a rudimentary VB application that uses the VMDB COM bindings and the ActiveX MKS.
- The VMDB infrastructure is developed sufficiently so that other application modules can take advantage of it and parallel development is possible.
  - VMDB library
  - VMDB-to-VMDB synchronization protocol
  - The uiVmdb module

Implementation strategy

- Allow applications to share state and expose functionality by creating instances of VMDB and synchronizing them via instances of the VMDB pipe abstraction
- Build each application's business and presentation layers so that they only interact with their local VMDB instance
- Implement several types of VMDB-to-VMDB pipes to accommodate these scenarios: intra-host, inter-host and host-guest communication

VMDB as a bus

- Not really an API but a bus
- Build components that can communicate on the bus (adapters)
- Build applications using the adapters and not using VMDB directly.



Main data structures: VMDB instance and VMDB Context

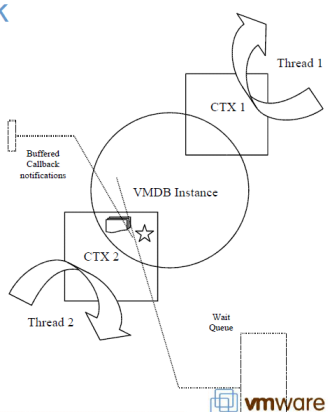
- The VMDB instance contains the current data and usually is one per process
- Accessing a VMDB instance is always done via a VMDB context, which keeps track of:
  - Buffered callback notifications
  - Transaction buffer
  - Current path
  - Username for the purposes of applying the VMDB security policy
- There can be many more than one context per process.

Threading model:

- The basic library does not expose an explicit thread abstraction.
  - Without the notion of threads, callbacks must be invoked manually by calling Vmdb_ProcessCallbacks.
- Threads need to be accounted for when:
  - Callbacks need to be automatically processed
  - Servicing a connection between VMDB instances
- Threads are accounted for by means of using IVmdbPoll.
- IVmdbPoll
  - An interface (a structure of function pointers and opaque state) to a thread's poll loop.
  - Exposes methods for registering callbacks with the poll loop but no method for running the poll loop.
  - May have different implementations. One such implementation is a wrapper on lib/user/poll.c

How callbacks work

- The context CTX 2 has callback registered on the path /a/b/c
- Thread 1 updates the value of /a/b/c in the VMDB instance via CTX 1.
- As part of the update, Thread 1 also stores an a callback notification in a buffer at CTX 2 and wakes up the wait queue at CTX 2. **Thread 1 does not execute the callback of CTX 2.**
- Assuming CTX 2 was registered with the poll loop of Thread 2, a poll-callback will fire as a result of waking up the wait queue and will call Vmdb_ProcessCallbacks for CTX 2.
- Consequently, the VMDB callback is executed asynchronously on Thread 2



Thread and process safety:

- It is safe to access VMDB via VMDB contexts on different threads
- However, one context should only be accessed by one thread at a time.
- The VMDB instances, VMDB contexts and their constituent data structures are safe to be memory mapped in the address space of different processes. However this requires that VMDB be allocated on a fixed-size shared memory pool.

Reference:

AppsRestructuring4_e_vmdb__2002.pdf

vmdb2003.pdf

11 VMM

x86 Virtualization Challenges

Not Classically Virtualizable

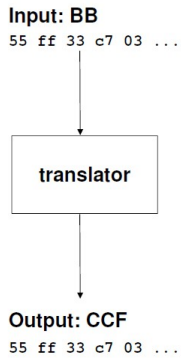
- x86 ISA includes instructions that read or modify privileged state
- But which don't trap in unprivileged mode

Example: POPF instruction

- Pop top-of-stack into EFLAGS register
- EFLAGS.IF bit privileged (interrupt enable flag)
- POPF *silently ignores* attempts to alter EFLAGS.IF in unprivileged mode!
- So no trap to return control to VMM

Deprivileging not possible with x86!

BT Mechanics



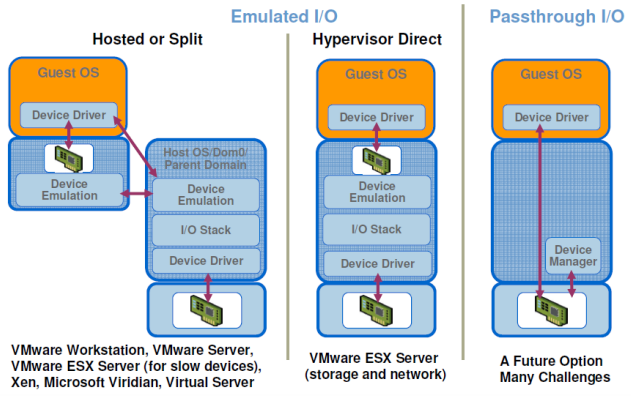
Each Translator Invocation

- Consume a basic block (BB)
- Produce a compiled code fragment (CCF)

Store CCF in Translation Cache

- Future reuse
- Capture working set of guest kernel
- Amortize translation costs
- Not "patching in place"

I/O Virtualization Implementations

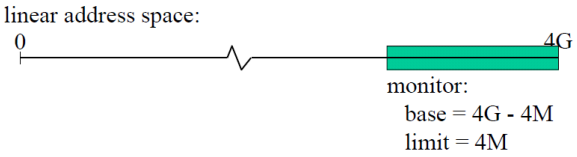


Reference:

Virtualization101.pdf

32 Basic

Monitor location



- Guest can use [0, 0xffc00000) freely
- Monitor virtual addresses: [0, 0x400000)

Segment truncation: keeping monitor "off limits" to guest

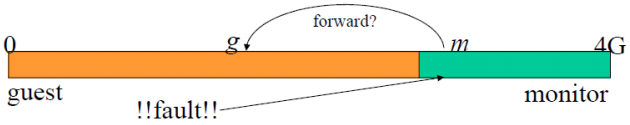
Cut guest segment limits at linear addr 0xffc00000

=> guest may fault on access that would have been ok
catch fault, perform access for guest
resume guest execution as if nothing had happened

=> slowdown

Fault and interrupt vector redirection: keeping monitor in control

guest tries to set a vector to addr. *g*
monitor overrides vector to m and remembers address *g*



if fault/interrupt happens, hardware jumps to *m*
inspect fault:
- absorb it (if induced by virtualization)
- forward to guest (if legitimate)

Execution engine

- Purpose of monitor: execute guest instrs
- Two ways to execute:
  - Direct execution (DE)
    - Just run the instructions
  - Binary translation (BT)
    - Translate the instructions
    - Execute translator output

But wait... ring compression?

- "Traditional" VMM:
  - Run guest privileged code at user-level
  - Priv. instructions trap
  - Trap handler: emulate the instruction

  } No BT
- x86 not *virtualizable:*
  - Some "priv." instr's don't trap at user level; instead they have different semantics
  - E.g.: user mode `popf`: no change to interrupt flag

Direct exec. vs. binary translation

- DE:
  - Minimal startup latency
  - Native speed
  - Not always possible
  - Typically used for user-level code (non-priv.)
- BT:
  - Translation overhead
  - Usually slower execution
  - *Must* handle all cases
  - All remaining cases, including kernel code (privileged)

To execute:

- If direct execution possible (and desirable):
  - Load virtual CPU state into physical CPU
  - Jump to guest code
- Else use binary translation:
  - Obtain (existing or new) translation for next block of guest instructions
  - Set up physical CPU for BT execution
  - Jump to translated code

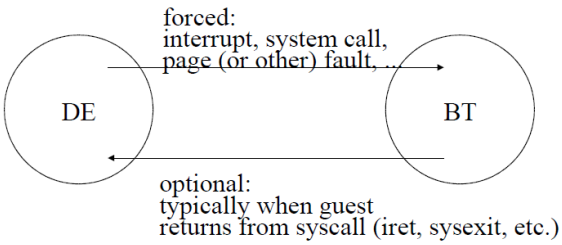
When is DE impossible?

- Privileged code (including real mode)
  - Prevent guest from executing priv. instr's that access HW: BT "neuters" the code
  - Confine guest's access to memory (sand box)
  - Example:
    - STI -- disable interrupts
    - Translate into code that clears *virtual* intr flag
- Certain descriptor table configurations
  - Cached descriptors (manifest only on CPU)

When is DE undesirable

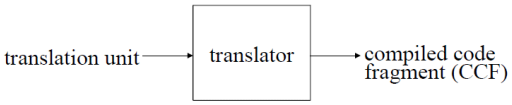
- Some instructions may create hidden faults due to segment truncation (or other effects of running in virtual machine)
- Absorbing HW faults is costly
- BT can generate non-faulting translation (e.g., optimized for high-memory access)

Transitions



Details: http://vmweb/~kma/syscall/syscall.html

Binary translation



- Each invocation of translator:
  - Consume one translation unit
  - Produce one CCF
- Translate on demand (dynamically)
- Interleave with execution
- Store CCFs in **translation cache** (TC) for future reuse

21

Translation: identical case

- Identify instructions that require no change
- Translate by "mem-copy"

```
80304a69 push %ebp
80403a6a push (%ebx)
80403a6c mov  (%ebx), ffffffff
80403a72 mov  %edx, %esp
80403a74 mov  %esp, 81c(%ebx)
80403a7a push %edx
80403a7b mov  %ebp, %eax
80403a7d call 80460ba4
```

Translation unit

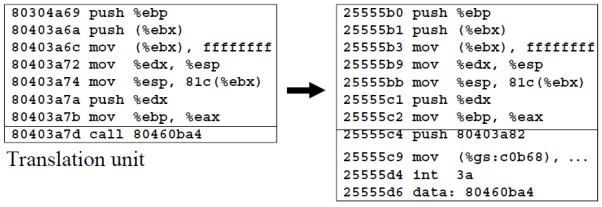

```
25555b0 push %ebp
25555b1 push (%ebx)
25555b3 mov  (%ebx), ffffffff
25555b9 mov  %edx, %esp
25555bb mov  %esp, 81c(%ebx)
25555c1 push %edx
25555c2 mov  %ebp, %eax
```

CCF

Translation: call

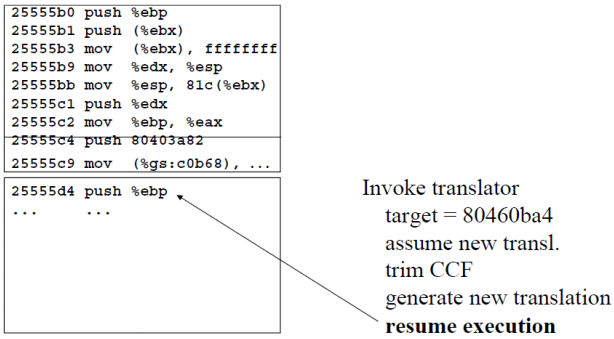
- Call instruction has two effects:
  - Change program counter (%eip)
  - Push return address on stack
- Preserve both effects across translation
  - Change program counter to *translation of callee*
  - Push *original* ("source") return address on stack
- Can't translate a call into a call
  - Would push wrong return address on stack

Translation: call



c4: push "source" return address
c9: establish hint for "return" instruction
d4/d6: invoke translator on callee

Execution of translated code



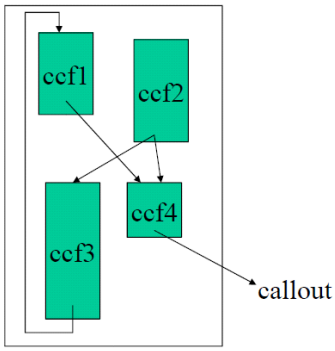
Other non-trivial translations

- All control flow - since translator relocates
- High-memory access
- In/out
- Privileged instructions (popf, lgdt, cli, movcr)
- Semi-privileged instructions (sgdt, rdtsc)
- Segment operations

Translator output: CCF

- Consecutive block of instrs
- Embedded and out-of-line data blocks
- May jump ("chain") directly to other CCFs
- May finish in "callout" to translator
- May "call out" to helper functions (e.g. I/O)

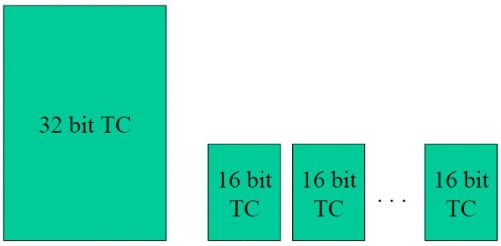
CCF management



CCF's live inside translation caches (TCs)

TC has data structures for finding existing CCF given "program counter" (really: Source Key)

TC management



Large, ~ 3 MB

64 KB each
Adaptively adjust # of these (0-8)
Distrib. code segm's according to volume

When TC full: flush all its code (no selective freeing today)

Modules: vmm/cpu/*

• directexec:	manage direct execution and faults in direct execution
• dt:	manage guest and monitor descriptor tables
• fpu:	floating point and SSE state
• idt:	fault, trap, and interrupt handling
• inout:	in/out
• mmu:	guest virtual memory, access to guest space from monitor
• priv:	privileged instruction emulation (CR registers, popf, cpuid, ...)
• segment:	emulation of instructions, interrupts and faults that change (code) segment
• singlestep:	controlled execution of instructions that access traced memory
• trace:	VMM service: allows tracking of read/write access to select guest locations
• vidt:	guest (virtual) interrupt descriptor table (compare idt.c)

Modules: vmm/bt/*

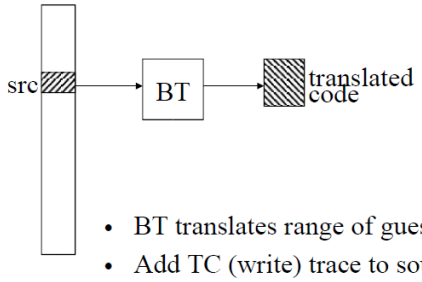
- annotation: places attributes on source instructions (e.g., performance hints)
- bt: BT control module ("glue")
- btpriv: translation of privileged instructions
- btseg: translation of instructions that modify segment registers
- tc_sync: synchronize VCPU to virtual instr. boundary upon fault or IRQ
- tc_coherency: track if guest writes to pages from which we have translated
- tc: translation cache module; manages all CCFs
- checkcode: handle transl's whose source code shares page with mutable data
- selfmod: handle self-modifying code (constant-modifying code)
- decoder: instruction decoder
- disassembler: 16 and 32 bit instruction disassembler (for debugging, logging)
- emulate: a variety of translations (e.g., most control-flow)
- farops: translation and execution of far calls, far returns, irets, long jumps
- linker: code generation utility for managing references to shared routines in TCs
- shared_code: generation of shared routines in TCs
- relocate: efficient translation of instr's that access high memory
- simulate: slow but general transl. (and exec.) of instrs that tend to fault on memory
- vphash[16]: mechanism for handling indirect control-flow transfers; hash table: eip->tca
- dispatch: infrastructure for calling out of (and reentering) translated code
- syscall: efficient transition between binary translation and direct execution
- pmatch: pattern matcher; generates faster translations of fixed instruction sequences

46

Intervention

- Interrupts (asynchronous or synchronous):
  - point interrupt vectors into VMM
  - conditional forwarding to guest (or host)
- Faults (and traps)
  - similar to interrupts, VMM gets control first
- Traces
  - detect guest access to address ranges

Example: TC trace



- BT translates range of guest instructions
- Add TC (write) trace to source range
- If source is modif. => notification
  => invalidate translated code

Trace types

- Basic types:
  - Read trace => notification on read access
  - Write trace => notification on write access
- Specific types:
  - TC: write (so BT can track code modification)
  - DT: write (details later)
  - MMU: write (details later)
  - VGA: read (for simulation of VGA display)
  - SVGA (obsolete), IOAPIC, APIC, ROM, IDT, VLANCE, COW

Example: Memory Mapped Devices

- Some devices are mapped to physical memory regions
- Apply appropriate traces on these regions
- Callback routine emulate device operation
- Future:
  - Cleaner interface to map/unmap a device or read/write device

Trace implementation

- Use page protection (read/only or unmap) to catch access attempt
  - Smallest unit that can be traced: 4 KB page
- Access attempt => page fault
  - Lift protection
  - Single-step the access
  - Restore protection
  - Notify trace clients (“callback”)

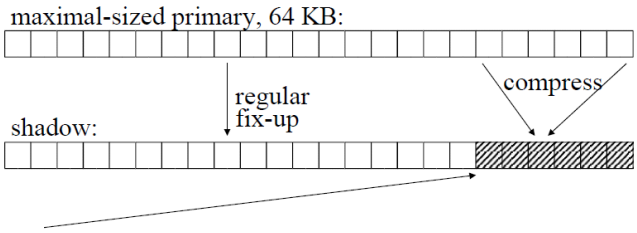
Traces attach to "physical" pages

- Page can be mapped in multiple locations
- Trace will fire on access through any of the page's locations
- Traces reflect "contents" of memory, not location of memory

x86 segmentation definitions

- *Segment:* a memory-mapped structure
  - Example: a code segment is a memory interval containing executable code [base, base + limit]
  - Other examples: data segment, call gate, task, LDT
- *Descriptor:* a 64 bit record that describes a segment
- *Descriptor tables* (LDT, GDT): arrays of descriptors
- *Selector:* a 16 bit integer that identifies a descriptor (by indexing into LDT or GDT arrays)

Sharing the GDT



Last ~20 entries reserved for
- monitor's use (flat segment, code segments)
- juggling guest descriptors that have no other "home" (cached segments, ones that cannot be shadowed)

The x86 MMU

- Paging based isolation is in addition to that provided by segmentation
- Hardware managed TLB
- Software invalidates a single entry or the whole TLB.
- CR3 points to the page root, CR2 is the faulting linear address
- 4K, 2M or 4M sized pages.

Page Table Cache

- 4 way set associative, 1024 lines, lru
- Key for the hash/search is the ppn of the page table
- Each cache entry contains
  - The mpn of the shadow page table page
  - The ppn of the virtual page table
  - Other flags : stale, is shadow of a root etc.

Validation Process

- Starts at Int14_PF (true, hidden, trace, or both)
  - Hidden #PF => translation is present in the virtual page tree, but not in the shadow
- A new shadow page table may be created
  - Trace placed on the virtual page table
- A new shadow pte is inserted
  - Shadow pte contains MPNs
    PMAP_GetMPN(PPN(virtual pte)) == MPN(shadow pte)
  - Accessed bit

Maintaining Consistency

- Accomplished by tracing the guest page table
- When virtual pte is modified
  - Shadow pte can be
    - Zeroed out and validated lazily (with a hidden #PF), or
    - Re-validated immediately
  - Trace can be retained or removed
    - High cost of trace faults (conditional simulation helps!)
    - Page may need to be totally refreshed.

PMAP, MonTLB & Zapping

- PMAP(ppn) = (mpn, traces)
- MonTLB(lpn) = (ppn, mpn, traces)
- Zap: when PMAP trims a ppn,
  - All the shadow pte's & montlb entries with the same *mpn* are invalidated
- Zap: when PMAP is instructed to change the traces
  - R/W bits in the shadow pte
  - Trace bits in the montlb entries.

Segmentation

- Mostly gone from long mode
- fs and gs: use as extra base register (NT compat!)
  - `mov r15, (fs:rbx)` ; really fs.base + rbx
  - fs: thread-local store
  - gs: kernel data
  - no limit checks
- cs: attributes only
- ds, es, ss completely gone

Reference:

monitor-talk_2001.pdf

monitor2_2001.pdf

64bit_VMM--2004.pdf

33 Memory Virtualization

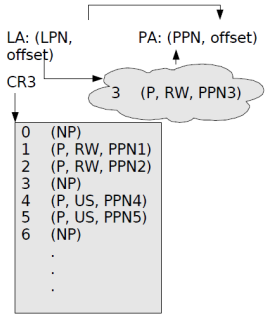
overheads

Guest writes to guest page tables need to be trapped So that shadows can be suitable modified

Guest writes to CR3 register need to trapped So that VMM can switch to the new set of shadows

Page faults need to be handled by the VMM So that “spurious” page faults are not forwarded to guest

MMU hardware: Page table update



- OS modifies the page table
- Accesses might still see old page mapping (no snooping!)
- TLB needs to be flushed:
  - CR3 reload
  - INVLPG

Shadow page tables

- Guest page tables store PPN and access permissions
- We need to remap pages: PPN->MPN
- We might restrict access permissions:
  - COW
  - Migrate
  - Traces
  - Hardware bugs
- -> Monitor maintains shadow page tables

Hidden page fault (2)

- Monitor #PF handler is executed
- Handler looks up guest page table
- Handler checks permissions
- Handler updates shadow page table
  - new MPN (derived from PPN->MPN mapping)
  - new permissions
- Handler resumes execution of the faulting guest instruction

Caching MMU: Trace hit (2)

- Guest tries to update its page table (mapped at LPN1)
- Page table is trace protected: shadow LPN1 mapping is not writable
- Monitor #PF handler is executed
  - Handler checks guest permissions for LPN1
  - Handler notices LPN1 has an MMU trace
  - x86 interpreter single steps guest instruction
  - Handler updates the shadow page table

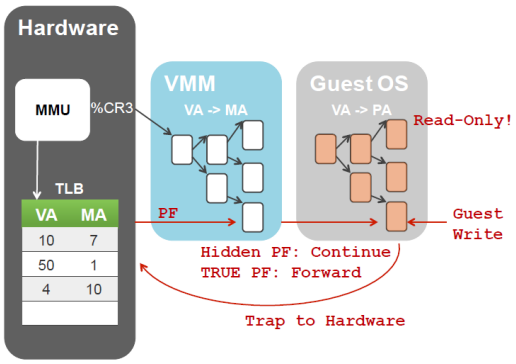
Caching MMU: Inline trace

- There is only a small number of guest instructions that modify guest page table
- Trace handler identifies these instructions
- Binary translator creates special translation for these guest instructions
  - No need to run x86 interpreter
  - No hardware page fault (use monitor-only writable mapping of the guest page table)

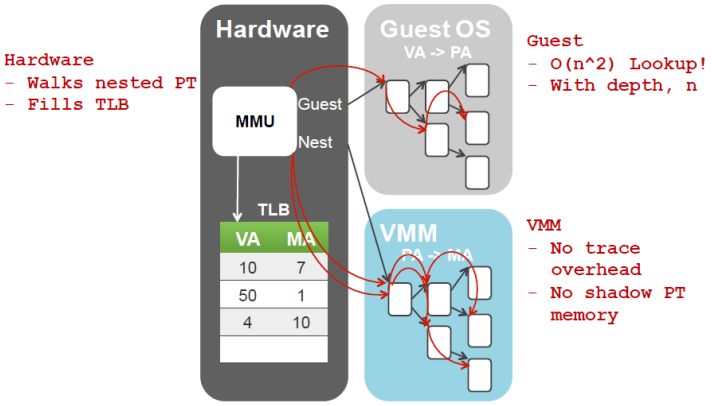
Hybrid MMU: Update (5)

- Most page tables are traced
- When the guest wants to update a page table, we drop the trace and mark the shadow as potentially stale. We then resume guest execution.
- Guest can now update as many entries as it wants in the page table
- When we need to synchronize shadow page table, we compare guest page against copy coherency page.

MMU Virtualization: Shadow Page Tables



MMU Virtualization: Nested Paging



Reference:

http://engweb.eng.vmware.com/techtalks/hybridMMU.pdf

2007-10-12-vivek-pandey-slides-000.ppt

34 BT

Why is BT used Over Hardware Assist?

Binary Translation is faster:

- Converts traps to callouts. Callouts faster than trapping
- Faster emulation routine
- Avoids callouts entirely

35 HV

Reference:

Virtualizing Hardware-Assisted Virtualization.pptx

36 SMP

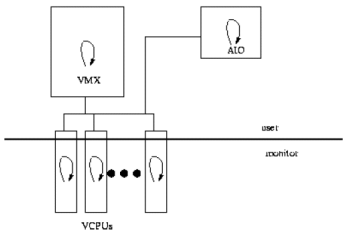
State of the vmm

- There is *still only one monitor*
- Multiple VCPU threads
- Concurrent programming model
  - Some state shared: pmap, mmu
  - Some state per VCPU: BT, DT
- Locks control concurrency

State of the vmx

- Single vmx thread
  - "assistant for VCPUs and devices"
- Polls for service requests
  - vcpu user calls ("RPC")
  - device interrupts
  - time queues
- Limited concurrency

VCPU user call multiplexing



- vcpus send requests to vmx
- vmx performs operation *on behalf of* vcpu
- vcpus wait for response

6

Deadlock prevention: lock ranks

- All locks have a rank
- Threads can acquire locks in ascending rank only
- Implication: no deadlocks
  - Either "lock(a); lock(b)" or "lock(b); lock(a)" produces rank violation
  - or both, if "rank(a) = rank(b)"

Single vmx thread: new kind of deadlock

- vcpu-0:
  - lock(a)
  - ...
  - usercall
- vmx thread:
  - poll
  - device service
  - lock(a)

- vmx thread needs lock to make forward progress
- vcpu thread will not release lock until user call completes
- user call never completes because vmx thread is blocked

Reference:

before_after_smp-talk_2002.pdf

37 Crosstalk

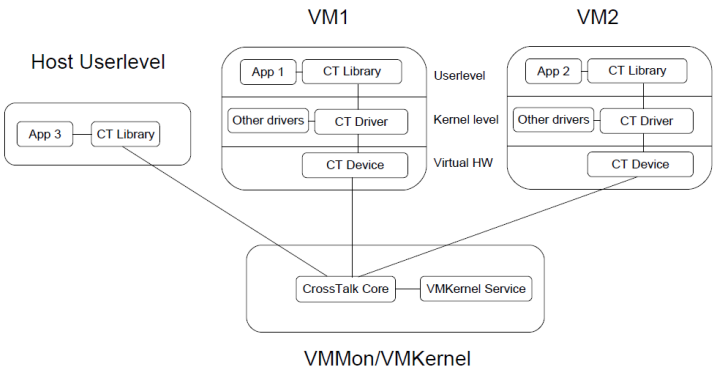
Overview

The CrossTalk framework provides a simple way:
- for two VMs to communicate with each other
- for a VM to communicate with an application on the host
- to implement high speed virtual IO devices

The framework consists of:
- a virtual device providing a simple call mechanism in and out of a VM
- a Datagram API to exchange small messages
- a Shared Memory API to share data
- an Access Control API to control what a VM can see
- a Discovery Service for publishing and looking up resources

CrossTalk will be available on both hosted and vmkernel/ESX based products.

CrossTalk Architecture



CrossTalk Device

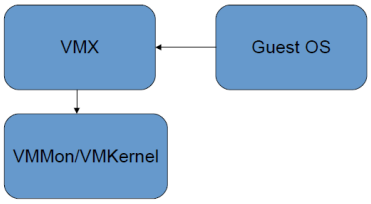
- PCI device
- 6 registers and an IRQ line
- Device backend implemented at user level

Hypercall
Guest call
Control
Status
Interrupt Cause
Interrupt Mask

CrossTalk Register Layout

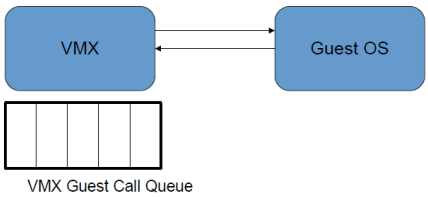
Basic call mechanisms: Hypercall

- Guest to hypervisor communication
- Rep out on hypercall port
- Returns success or failure
- The call could be asynchronous
- Call format:
  - Vector
  - Call size in bytes
  - Call data

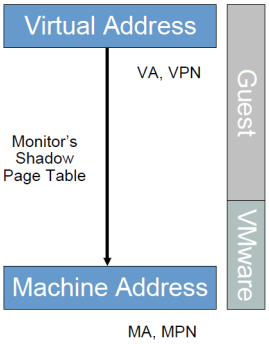


Basic call mechanisms: Guest call

- Hypervisor to VM communication
- Calls to a given VM are queued in the device backend
- Guest is interrupted
- REP IN on guestcall port to read call data.



Hosted memory management: Terminology

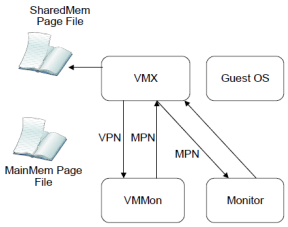


- Guest accesses a given VPN
- Monitor's page fault handler is called
- Monitor walks guest page table to find the corresponding PPN
- Monitor asks VMX for the corresponding MPN
- When a VPN, MPN entry in the shadow page table is valid, the MPN must be locked on the host.

Hosted memory management

- Guest physical memory is backed by a page file on the host.
- VMX may partially map its contents into its address space at a given time.
- When monitor instructs VMX to lock an MPN corresponding to a PPN:
  - VMX maps in a portion of the page file containing this page
  - Sends a request to vmmon to pin the page on the host
  - VMX unmaps the page (in theory)
- When the monitor decides to unlock an MPN the same process is followed.

Guest Access To Shared Memory



1. Guest accesses VA
2. Monitor PF handler looks up PPN
3. Asks VMX to lock corresponding MPN
4. VMX determines it is a shared page
5. Maps it in from shared mem page file.
6. Asks VMMon to lock the corresponding MPN

CrossTalk Userlevel API Implementation

- Each userlevel process is represented by a CrossTalk process with its own file handle to poll for and read data from the CrossTalk kernel driver.
- Accesses the kernel API via CrossTalk Driver IOCTL interface
- Provide a userlevel library wrapping the IOCTL interface and doing initialization required for the process to use CrossTalk
- Access is restricted by normal file permissions and not by CrossTalk. I.e if an application can open the CrossTalk device it gains access to all of the crosstalk API we expose via the IOCTL interface.
- We have currently exposed the Datagram via the IOCTL interface and written wrapper functions identical to the kernel level Datagram functions.

38 BIOS

Reference:

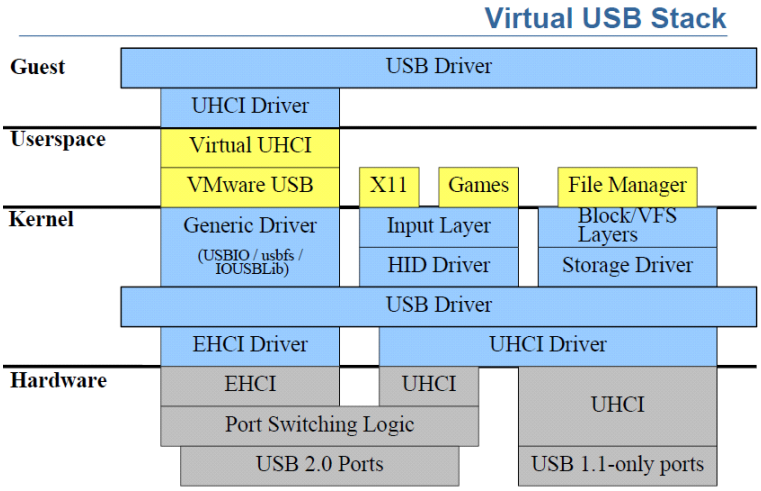
2007-11-02-adhyas-avasthi-slides-000vChipset and vBIOS HotPlug Initiatives.ppt

Reference:

2007-03-02-dmitriy-budko-slides-000.ppt

2009-05-08-EFI-slides.ppt

39 Virtual Device

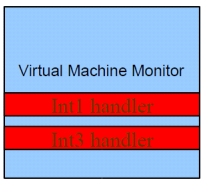


Reference:

http://engweb.eng.vmware.com/techtalks/USB_tech_talk-7.pdf

40 Debugging

Virtual Machine Monitor



- Int1 handler (trace instruction)
- Int3 handler (breakpoint)
- Some new data structures to help us find our way

Reference:

debugging_vmx2001.pdf

Monitor core debugging example

```
- Use 'SearchForPat' to find PTEs with the target value:
# ./corequery open vmmcore-vcpu0 \; searchforpat 0xdeadbeef 0xffffffff \;
Found Pattern 0xdeadbeef in anon page 0x18f628 at offset 0xa8c
Found Pattern 0xdeadbeef in anon page 0x1d2887 at offset 0x864

- Dump monitor page tables (MON_PAGE_TABLE:0xfffffffffc01c:0x20) and search:
(gdb) x/16384xg 0xfffffffffc01c000

# grep 18f628 pt.txt
0xfffffffffc01d920:     0x000000018f628061      0x000000011c7a8061

-Calculate mon VPN / mon VA for the PTE:
(gdb) p/x (0xfffffffffc01d920 - 0xfffffffffc01c000) / 8
$7 = 0x324
(gdb) p/x 0xfffffffffc000000 + (0x324 << 0d12)
$8 = 0xfffffffffc324000
(gdb) x/xw 0xfffffffffc324000 + 0xa8c
0xfffffffffc324a8c <FPU_PrepareX87PatchingForReplay+412>:       0xdeadbeef

-Repeat:
# grep 1d2887 pt.txt
0xfffffffffc01dcb0:     0x00000001d2887061      0x00000001eddb1061

(gdb) p/x (0xfffffffffc01dcb0 - 0xfffffffffc01c000) / 8
$9 = 0x396
(gdb) p/x 0xfffffffffc000000 + (0x396 << 0d12)
$10 = 0xfffffffffc396000
(gdb) x/xw 0xfffffffffc396000 + 0x864
0xfffffffffc396864 <HVExit+1300>:       0xdeadbeef
```

Debugging

- Tracking structures:
  - Page tracker/PMIO ring useful for debugging memory corruption
    - Hosted only *
  - Ring buffer in the monitor useful for recent VM execution history and instrumentation
  - CCFs in BT mode provide record of recently executed guest code
- Questions/discussion ???

Reference:

16-debugging.pdf

vmm-crash-hang-for-gss.ppt

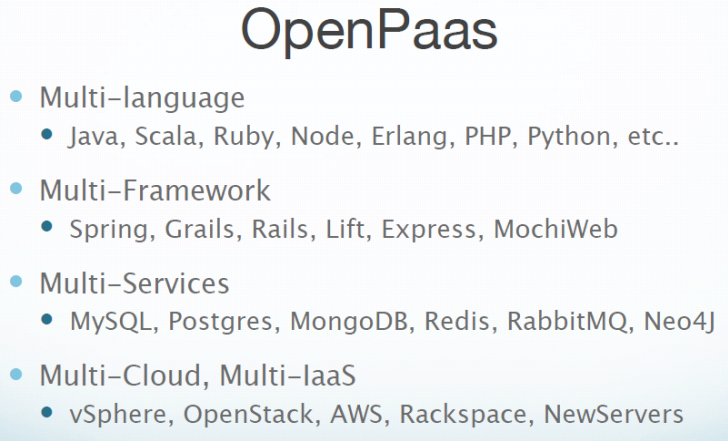
vmm-for-gss.ppt

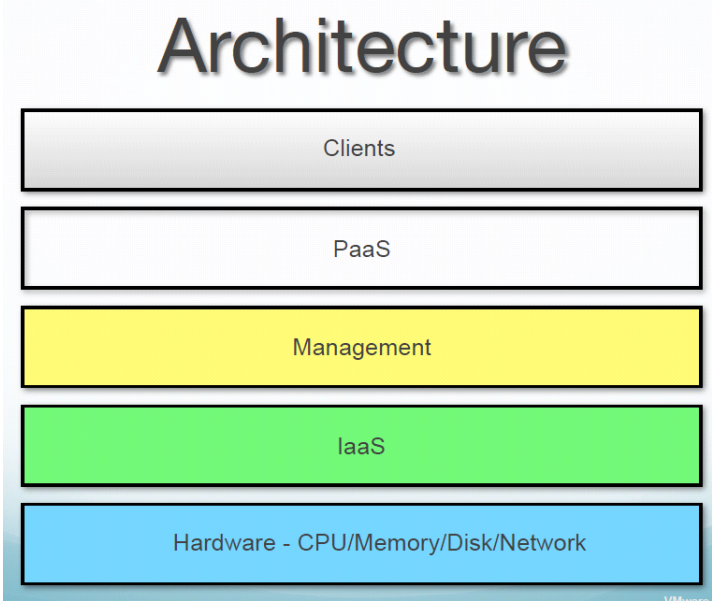
11.1.1 **Stats**

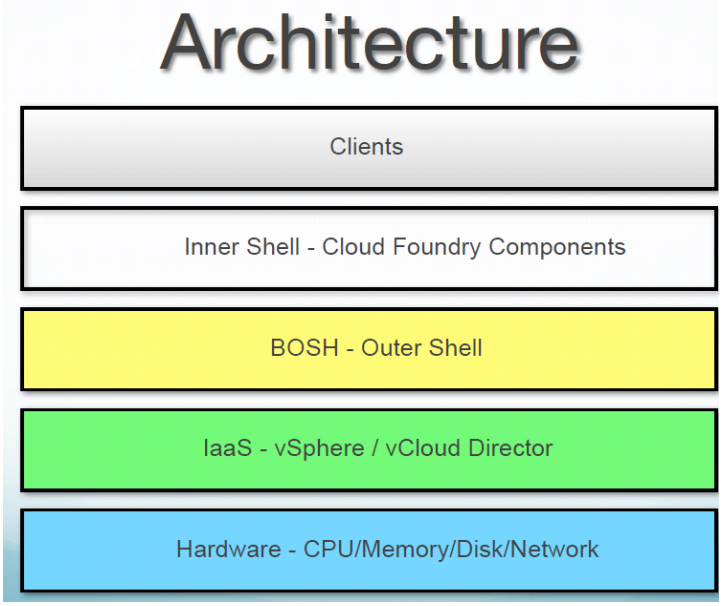
Reference:

ari_stats_overview_talk_monitor_stats.ppt

12 Cloud Foundry



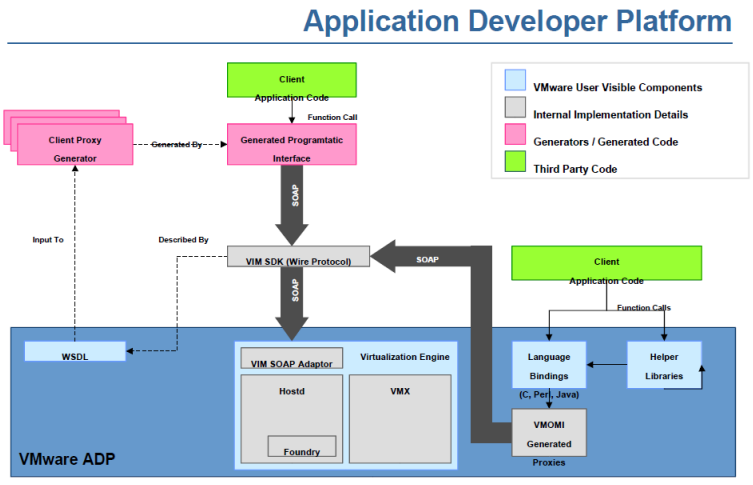




Reference:

CF_TechTalk.pdf

13 ViClient



Reference:

BootCamp-VIM-VIClient.ppt

41 Hostd

VIM

42 VMRC

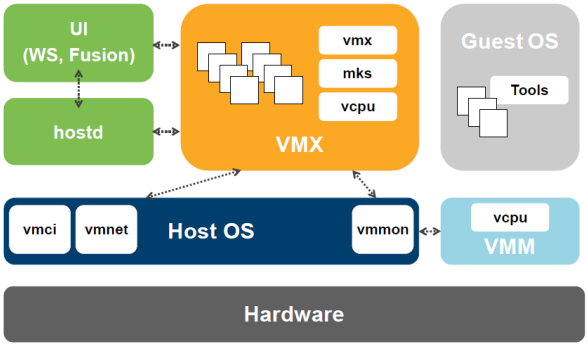
14 Hosted

43 Architecture

Hosted Products Overview

- **Workstation**
  - Power-user, full featured UI
  - Windows and Linux
- **Fusion**
  - Fewer advanced features
  - Mac OS
- **Player**
  - Basic features: Create and run VMs
  - Windows and Linux (free as in beer)
- **View Local Mode**
  - Offline sync of ESX VMs
  - Encrypted VMs
  - Windows

The Big Picture



VMX: Components

- **General Infrastructure**
  - Threads, locking, signals, stats, licensing
  - External interfaces (Vigor, VIX, VMHS)
- **Monitor Support**
  - vmmon, SharedArea, guest memory access, etc.
- **Virtual Device Infrastructure**
  - IRQ, I/O space & memory mapped I/O callbacks
  - Checkpointing
- **Virtual Devices**
  - Networking, storage, video, USB, etc.
- **Tools Support**
  - HGFS, DnD, etc.

VMX: External Interfaces

- **Vigor (VMX Interface Generated Object Representation)**
  - Auto-generated interface to the VMX binary
  - Auto-generated documentation (Javadoc), and unit tests
  - Non-versioned, non-public API
  - Under heavy development!
- **VIX (a.k.a Foundry)**
  - C-language client library (Perl, COM, Java bindings, vmrun)
  - RPC-style commands
  - Public (stable) and SemiPublic (unstable) APIs
- **VMDB + VMHS (old)**
  - Tree with ASCII strings; transactional
  - Distributed registry with notifications

VMX: VMX Thread

- **The initial thread in the VMX process**
  - It calls the “init” and “power on” methods of each subsystem
  - Starts all other critical threads (mks, vcpu)
- **Waits in poll() / select() after device and VMM initialization**
- **Poll loop fires callbacks**
  - Real time callbacks (“2 us by user's clock”)
  - Virtual time callbacks (“2 us by VM's clock”)
  - Device event callbacks (fd or HANDLE)

VMX: VCPU Thread

- **One per virtual CPU, each corresponds to thread in VMM**
  - Hosted: same thread via world switch
  - ESX: distinct threads, never concurrently scheduled
- **VCPU threads handle**
  - RPCs from VMM
  - Much (not all) device emulation
- **One lock serializes the VMX and VCPU threads**
  - The (Big) User Level Lock, BULL
  - Some high-performance paths have their own locks
  - Moving toward more fined grained locking!

VMX: MKS Thread

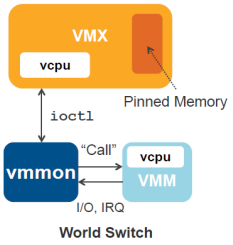
- **MKS = Mouse + Keyboard + Screen**
- **MKS thread handles**
  - Keyboard and mouse input
  - Graphical output
- **Avoids using the BULL as much as possible**
  - Input from interactive devices is latency sensitive
  - Graphics processing is expensive

VMX: Worker Threads

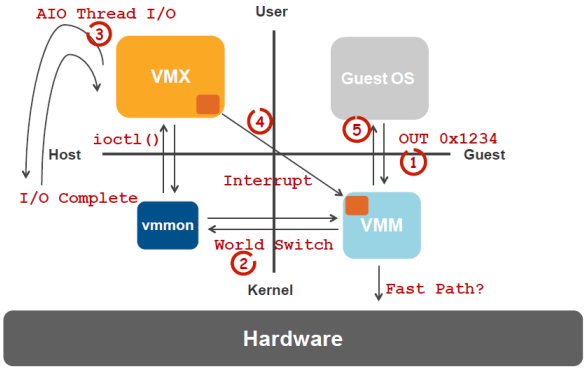
- **The VMX process has worker threads**
  - bora/lib/workerLib is a library available to any application
  - VMX is capped at 96 threads total
  - Used to perform operations asynchronously from the guest POV
- **Common clients include**
  - Storage Backend (e.g. Issuing reads and writes on a vmdk)
  - HGFS (e.g. Copying files between host and guest)
  - Background snapshots (e.g. Lazily copy guest's memory to disk)
  - Asynchronous screenshots of the guest

VMMon Driver

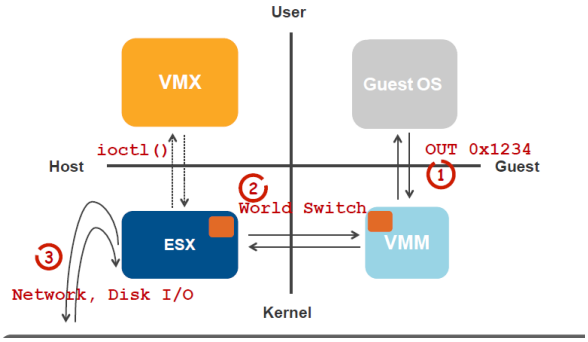
- **The VMX process uses it to transfer control between Host and Guest world (i.e. world switch).**
- **Reflects IRQs delivered in VMM back to the Host OS.**
- **Allocates / manages host RAM for VMM and VMX.**



Hosted I/O



Hosted I/O: ESX Compared

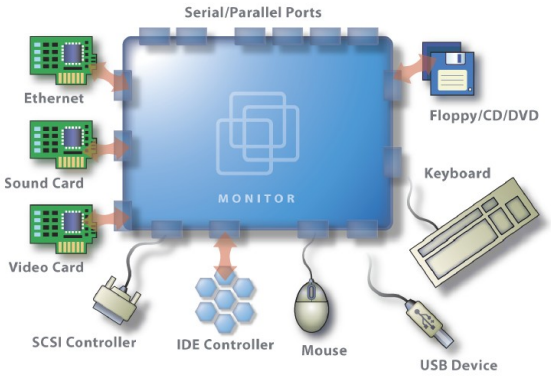


CPU Virtualization

- **Binary Translation (BT) + Direct Execution (DE)**
  - Unprivileged code (v8086 / CPL3) runs in DE
  - Privileged code runs with BT
  - BT can either 'call out' directly or trap and emulate / patch
  - DE must trap and emulate
- **Hardware Assist: VT / SVM**
  - Fewer traps, but each trap is more expensive
  - Performance gains variable
- **Trap Sources: CPU, MMU, I/O**
  - EPT / NPT reduce traps, but exhaust TLB
  - VT / SVM reduce traps, but traps are expensive

See *"Performance Aspects of x86 Virtualization"* by Ole Agesen.
https://wiki.eng.vmware.com/TechTalkArchive

Virtual Devices

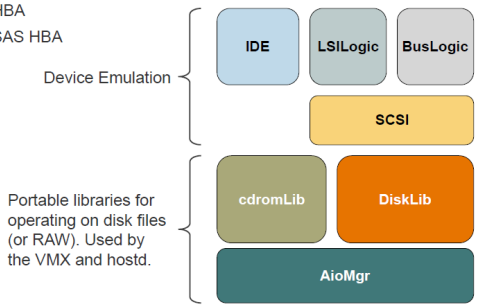


Chipset Virtualization

- **Mostly emulate 440BX/PIIX4 motherboard**
- **Some differences include**
  - Ours has an IOAPIC
  - Supports modern CPUs
  - ~100 PCI / PCIe slots
- **Firmware**
  - BIOS: Licensed Phoenix BIOS (highly modified)
  - EFI: Based on EDK2 with VMware device support
- **ACPI 3.0 support for guests**
  - Power-on suspend, hibernate, etc.
  - Hotplug (PCIe) devices, memory, CPU

Storage Virtualization

- **Four types of storage adapters**
  - BusLogic HBA
  - LSILogic HBA
  - LSILogic SAS HBA
  - IDE

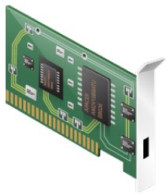


Storage Virtualization

- **DiskLib supports different kinds of disks**
  - Flat: preallocated
  - Sparse: grows as the guest writes
  - Device: raw disks **DANGER!**
- **Usually just a file on the host's disk**
- **DiskLib implements COW**
  - REDO logs or delta disks
  - Used for creating snapshots
- **All I/O is done asynchronously**
  - VMX worker threads do the blocking

Network Virtualization

- **Four types of NICs**
  - vlance based on the AMD lance chip
  - vmxnet paravirtualized version of vlance
  - Intel e1000 and e1000e
  - vmxnet3 passthrough-capable device
- **The "Split device"**
  - Most of the networking code is in the VMX process
  - Some performance critical parts are in the monitor
- **The e1000 supports certain kinds of offloading**
  - IP & TCP checksum offloading
  - TCP Segmentation offloading, a.k.a. TSO
- **The lance adapter can morph into a vmxnet adapter**



Graphics Virtualization

- **The MKS thread performs most graphics device emulation**
  - VGA adapter
  - SVGA adapter
- **Custom SVGA chipset plus VBE 2.0 BIOS**
  - Paravirtualized so we have to maintain our own guest video drivers
  - 2D primarily done in software by the guest
- **Rendering on the host uses host APIs!**
  - X11, GDI, D3D, OpenGL, VNC
- **3D in the guest**
  - DirectX 9.0c, Shader Model 2.0
  - OpenGL in Windows guests (Linux soon!)

Passthrough Devices

- **Good strategy for certain peripherals**
  - USB device level passthrough
  - CDROM supports a mode called "cdrom-raw"
  - SCSI generic
- **Reduces the VM portability!**
  - All passthrough devices unplugged at snapshot/VMotion

Resources

- **Reading**
  - Virtualizing I/O Devices on VMware Workstation
    - https://www.vmware.com/pdf/usenix_io_devices.pdf
  - GPU Virtualization on VMware's Hosted I/O Architecture
    - http://www.usenix.org/events/wiov08/tech/full_papers/dowty/dowty.pdf
  - A Comparison of Software and Hardware Virtualization
    - http://www.vmware.com/pdf/asplos235_adams.pdf
  - AMD-V™ Nested Paging
    - http://developer.amd.com/assets/NPT-WP-1%201-final-TM.pdf
- **Tech Talks**
  - MTS Tech Talks
    - https://wiki.eng.vmware.com/MainTechTalks
  - VMKernel Tech Talks
    - https://wiki.eng.vmware.com/VMkernelTalks

Reference:

Hosted_Architecture_070912.pdf

https://www.vmware.com/pdf/usenix_io_devices.pdf

[GPU Virtualization on VMware's Hosted I/O Architecture]

http://static.usenix.org/events/wiov08/tech/full_papers/dowty/dowty.pdf

[A Comparison of Software and Hardware Techniques for x86 Virtualization]

http://www.vmware.com/pdf/asplos235_adams.pdf

44 OVF

45 P2V

1 Remoting

3 basic layers in a hosted UI

cui – library which abstracts platform

lui/wui (and dui) – platform-specific library which performs some of the work of mapping UI to cui

vmui (or player, or vmrc, or fusion) – native UI implementation for a specific product

below that is bora/lib utilities, vmhostsvcs (and Vix/Foundry)

communication to running VMX happens over VMDB pipe, foundry automation socket, and other domain-specific communication channels (mks, tools socket, backdoor, etc).

Reference:

Remoting in Workstation 8.pptx

15 Virtualization Future

Summary

- Virtualization will eliminate complex hardware management and broad application support
- VMM is the center of universe

16 Checkpoint

46 Basic

What is checkpointing?

- Saving and restoring the state of a running virtual machine
  - CPU, memory, disk
  - motherboard resources
  - I/O controllers
- Challenges and Limitations
  - identifying all the state that needs to be saved/restored
  - quiescing the VM
  - pass-through device state
  - removable media
  - time-sensitive state
  - forward/backward compatibility

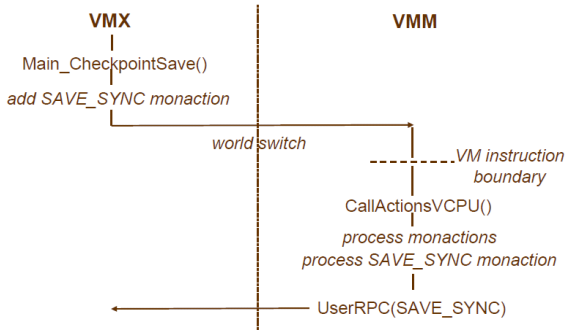
State to be saved

- Brute force approach - complete process dump
  - not portable - across releases, across platforms
  - VMM and VMX state present in checkpoint
- Rule: save only VM state, no implementation state
  - Virtual CPU state - vcpu
  - VM memory - mapped memory regions
  - Disk contents - COW disk file
  - Motherboard and I/O Devices - all registered virtual devices
  - Modified configuration variables

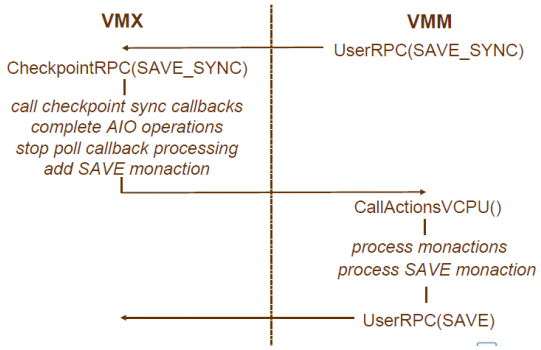
Saving a checkpoint

- Initiate a checkpoint request
- Quiesce VM state, call sync callback functions
  - Bring the VCPU to an instruction boundary
  - Drain all AIO operations
  - Handle all outstanding monitor actions
- Save VM state, call checkpoint callback functions
- Continue execution or Power off

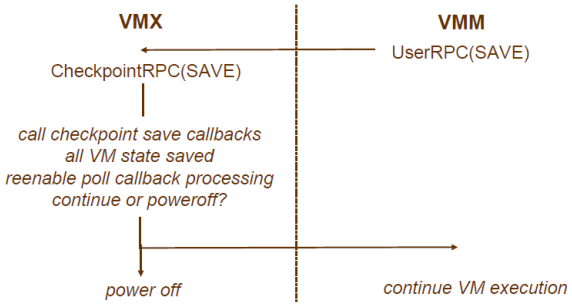
Checkpoint save protocol (initiate)



Checkpoint save protocol (sync)



Checkpoint save protocol (save)



Restoring a checkpoint

- Init and PowerOn functions will set default values
- Checkpoint callback functions will restore state
- VM state
  - restore from checkpoint file
- VMM and VMX state
  - compute and regenerate any non-default values from VM state

How to program it

- Register sync and checkpoint callback functions
  - via Checkpoint_Register
  - via Device_Register
- Checkpoint callback functions
  - API derived from SimOS
  - state saved as typed <tag, value> items
  - order of saved state does not matter
  - identical call to save or restore an item
  - independent of other callback functions
- See http://geneva/~mts/WebSite/notes/checkpoint_instructions.txt for checkpoint programming API

Registering Callbacks

1. Checkpoint_Register(name, cptcallback, synccallback, userdata);

```
Checkpoint_Register("cpu", CheckpointCPUCallback,
                    NULL, vm);
```

2. via Device_Register(vm, device_header);

```
devhdr.name = "Floppy";
devhdr.sync = FloppySync;
devhdr.checkpoint = FloppyCheckpoint;
Device_Register(vm, devhdr);

Checkpoint_Register(devhdr.name, devhdr.checkpoint,
                    devhdr.sync, devhdr);
```

Checkpoint Programming "Do"s and "Don't"s

- Do
  - keep checkpoint callbacks up to date
  - use version numbers
  - preserve backward compatibility
  - read http://geneva/~mts/WebSite/notes/checkpoint_instructions.txt
- Don't
  - rename tags
  - change an item size
  - add VM state without providing checkpoint callback
- Tools
  - struct size checker in DEVEL builds
  - checkpoint file decoder in bora/support/checkpoint/decodecpt.c

Applications of checkpointing

- Suspend/Resume
- Repeatable Resume
- General checkpointing
- Load balancing
- High-availability
- Backups
- Debugging

Reference:

checkpoint_2001.pdf

10-checkpoint.ppt

Reference:

2008-05-16-kevin-christopher-slides-000Virtual Machine Checkpointing .ppt

cptcompat_checkpoint_-2003.pdf

17 Tools

Tools Apps

Component	Vista+		
	Install	Upgrade	Uninstall
Core ("Tools Apps")	OK W2K8R1 disk reg setting	OK	OK
Display - XPDM	reboot	reboot if changed	reboot
	Win7/W2K8R2+: N/A		
Display - WDDM	pre-Win8: reboot Win8: OK	OK	pre-Win8: reboot Win8: OK
Mouse - PS/2	reboot	reboot if changed	reboot
Mouse - USB	OK	OK	OK
buslogic/pvscsi - boot device	reboot	reboot if changed	not uninstalled
buslogic/pvscsi - non boot device	OK	reboot if drive(s) are in use	reboot if drive(s) are in use
vlance/vmxnet2/vmxnet3	OK	OK	OK
Memory Balloon	OK	OK	OK
vss / sync	OK	OK	OK
vsock/vmci	OK	sockets in use	sockets in use

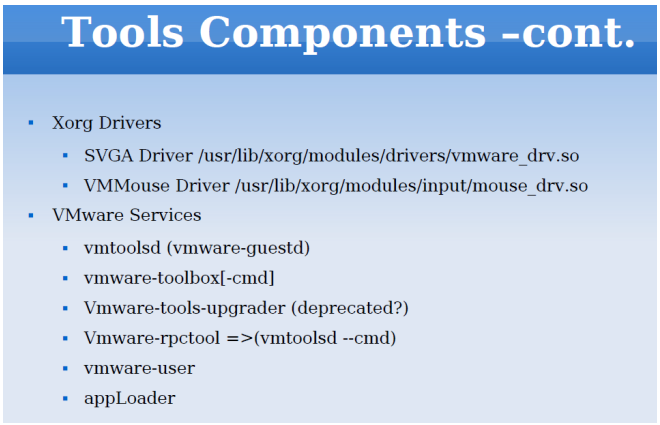
Reference:

minreboots.pdf

Tools Components

```
#service   vmware-tools start
Switching to guest configuration:                    done
  Paravirtual SCSI module:                           done  --pvscsi
  Guest filesystem driver:                           done  --vmhgfs
  Mounting HGFS shares:                              failed
  Guest memory manager:                              done  --vmmemctl
  Guest vmxnet fast network device:                  done  --vmxnet
  VM communication interface:                        done  --vmci(crosstalk)
  VM communication interface socket family:          done  --vsock
  Blocking file system:                              done  --vblock
  VMware User Agent (vmware-user):                   done  --vmare-user
  File system sync driver:                           done  --vsync
  Guest operating system daemon:                     done  --vmtoolsd
  Virtual Printing daemon:                           done --

  //Driver for the VMXNET 3 virtual network card:  - vmxnet3
                                                   -- vmdesched #deprecated
```



vmware-tools-cli.pdf

20091126_tools_foundry_intro.pdf

47 VMCI

Reference:

2008-11-21-jorgen-hansen-slides-000.ppt

http://www.vmware.com/products/beta/ws/VMCIsockets.pdf

18 Windows

DOS-based releases
- Windows 1.x
- Windows 2.x
- Windows 3.x / WFW 3.x
- Windows 95
- Windows 98
- Windows Me

NT-based releases
- 3.x: Windows NT 3.1 / 3.5 / 3.51
- 4.0: Windows NT 4
- 5.0: Windows 2000
- 5.1: Windows XP
- 5.2: Windows Server 2003 (and 64-bit XP)

Next major "dot oh" release of Windows
- 5.0: Windows 2000
- 5.1: Windows XP
- 5.2: Windows Server 2003 (and 64-bit XP)

- 6.0: Windows Vista

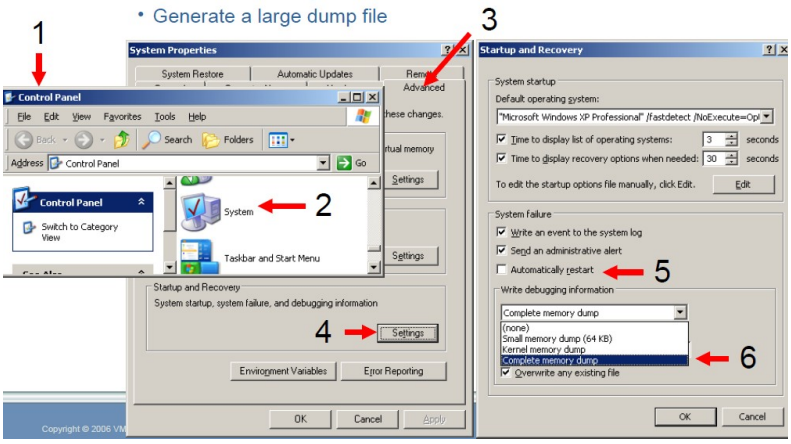
Vista is the successor to Windows XP
- Scheduled release is 2006Q4 / 2007Q1 / 2007Q2

Windows "Longhorn" Server
- Successor to Windows Server 2003
- ETA: a year or two after Vista

48 Debugging

How everyone should configure their box

Configure system using Control Panel
- Don't automatically reboot
- Generate a large dump file



What to include when filing a host/guest BSOD PR

- Include the memory dump file
  - This is the most vital piece of information, followed by VM's config file and log file (also vmkernel log for esx)
    - Minidumps files are stored in c:\windows\minidump, and accumulate
    - Full memory dump is stored at c:\windows\memory.dmp
      - Subsequent BSOD will replace this file
      - File is created on next reboot – wait for file to be created (e.g., wait several minutes, and/or until file stops growing)
  - The VMware "support script" isn't ideal – it just grabs a bunch of files on the host (but not the guest) and it's up to the recipient to figure out what VM failed and which files belong to that VM
- Ideal: Try to determine what caused/provoked the crash
  - Steps covered later in this talk
  - Goal: know how to determine what component crashed the system, so can file PR under a category that's more specific than "host bluescreen" or "guest bluescreen"
  - Feel free to let owner of category figure out what went wrong
  - If you are the owner and need more detailed info, there will be a 2nd talk

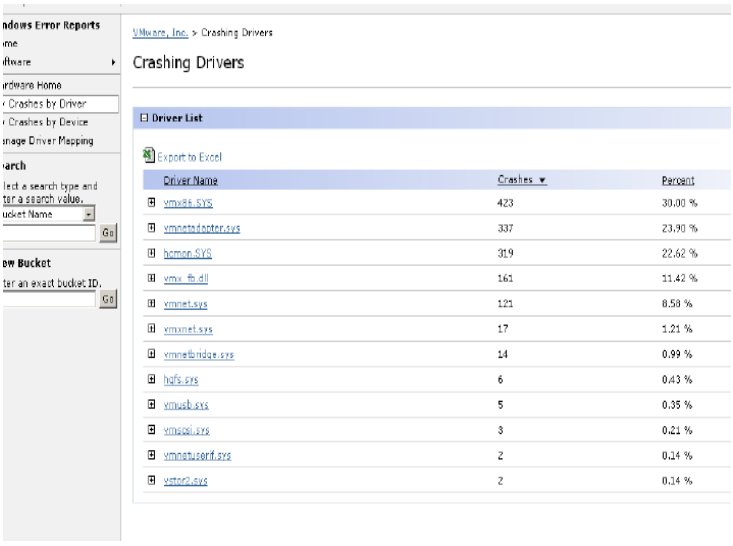
Windows Error Reporting (WER)

Lifecycle of an application crash (XP and later)

- A crash (e.g., fault or exception occurs) in an application
- Application might attempt to handle/capture crash
- Crash gets reported to kernel
- Code runs to capture crash state into a dump file
- Dump file and other information sent to Microsoft
- Microsoft analyzes stack dump
  - Application name, module name, version information, and faulting IP used to assign crash to a "bucket"
  - ISVs report names of their executable modules to Microsoft
  - Buckets are assigned to vendor based on who owns the module containing the faulting IP (vendor can see after 3 hits in bucket)
- Vendors obtain crash information from Microsoft
- Vendor fixes bug and issues update
- Optional: Vendor provides information that is shown to customers when they hit a crash belonging to the bucket

Something similar to the above occurs for BSODs

- Kernel minidump (64KB) is sent to Microsoft after system reboots



Debugger Options

Microsoft's debuggers
- GUI Debuggers
  - Visual Studio
  - WinDbg (often pronounced "WindBag")
  - Visual Studio and WinDbg don't share any code
- Text Debuggers
  - KD (kernel debugger)
  - CDB / NTSD (text version of WinDbg)
    - NTSD is included with windows
- This talk will focus on WinDbg
  - Usually better than Visual Studio, but not in all regards
  - Available for free on Microsoft's website
  - Easier to learn than the text-based debuggers
  - Can be used to debug both application and kernel code, so often the same commands apply in both contexts

```
[boot loader]
timeout=3
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /fastdetect /debug /debugport=com1 /baudrate=115200
```

Help windbg to help you

Windbg needs symbol files (.pdb/.dbg) to know how to report useful information
- For minidump files (e.g., 64KB) windbg needs the actual .exe/.dll/.sys executable file
- Windbg can use the files on your local PC, but doesn't help much if debugging a different version of Windows
- Can download and install symbol package from Microsoft:
  - http://www.microsoft.com/whdc/DevTools/Debugging/symbolpkg.mspx
- Easiest to configure windbg to use Microsoft's symbol server
  - Windbg uses the information it knows about a file to generate a unique HTTP request to Microsoft's server
  - File is then cached locally for future use
  - Symbol server has a superset of the symbol packages, since it also has files for the various patches and fixes
  - Server has symbols files for NT4 and later, but only has executable files for Windows XP and later
    - If debugging Windows 2000 minidump, might need to get the matching file from that PC, find another W2K PC with the same file, or try to get something "close" and tell windbg to ignore the mismatch

Reference:

windowsDebugging-2006.pdf

windows-debugging-presentation_2002.pdf

49 Partition & File System

Reference:

2009-03-27-Aleksey-Pershin-slides.ppt

19 Linux

50 Guest Issues

Reference:
guest-for-gss.ppt

51 KVM

#----------#
KVM Architecture.
#----------#

- KVM architecture allows for fast development, test cycles, larger ecosystem
- Host Hardware layer
  - ø Intel or AMD virtualization extensions must be enabled in the BIOS
- Kernel Mode layer
  - ø Linux kernel + KVM module
- User Mode layer
  - ø VMs are processes running alongside other processes on the system
  - ø QEmu for hardware emulation
  - ø libvirt is for toolkit/API, and for managing with virt-manager or virsh
- Is KVM a type 1 hypervisor?
  - ø Just because KVM co-exists with RHEL does not change its bare metal type
  - ø Type 1 hypervisors have direct access to the hardware
  - ø KVM provides direct access to HW with Intel/AMD virtualization extensions or with "virtio" for para-virtualization to physical NICs or hard drives
- Is KVM a type 2 hypervisor?
  - ø VM run as processes in the host OS - so it is no different then VirtualBox
  - ø Reality is who cares - KVM falls in the middle of the Type 1 and Type 2

#----------#

Comparisons Between RHEL KVM, RHEV, and vSphere.

#----------#

• Both Red Hat solutions are priced per processor (no licensing)

• RHEL KVM

ø Management level is virt-manager (GUI) or virsh (CLI)

ø Platform level is RHEL + KVM

ø Install footprint is as large as the components selected during install

• RHEV

ø Management level is RHEV Manager installs on a RHEL server

ø Platform level offers 2 models for host support

- RHEV Hypervisor (90% installations)
  - Minimal Linux installation with the KVM kernel module
  - Intended for only virtualization workloads (no user space support)
- RHEL as a Hypervisor (10% installations)
  - Full RHEL install with KVM
  - Used for environments who want script customization

• "Similar Enterprise Grade Solutions... but NOT identical"

#----------#

RHEV 3.1 Architecture.

#----------#

• A proven company for open source software in production environments

• Red Hat is a "glue factory" for open source and Linux technologies

• RHEV architecture is based on a collection of open source technologies

ø (1) RHEV Hypervisor

ø (2) RHEV Manager installed on a RHEL host

- RHEV Manager has transitioned from Windows framework to Linux framework
- Components like JBoss, PostgreSQL, ovirt-engine

ø (3) User Portal (mostly for RHEV for Desktops VDI customers)

ø (4) Web Admin Portal – for virtualization management

ø (5) Console (SPICE)

- Linux: yum package spice-xpi
- Windows: SPICE ActiveX control for IE

---

* Simple Protocol for Independent Computing Environments (SPICE)

Reference:

rhev31_TechTalk_Toby.pptx