Flexible Hypervisor

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFIMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Master of Engineering
IN
Computational Science

BY

Ishwar Raut

Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2015

DEDICATED TO

My Family

Acknowledgements

First and foremost, I express my sincere and heart-felt gratitude to my adviser and guide, Prof. S K Nandy and Dr. J. Lakshmi for their guidance and support throughout the coursework. The insightful discussion with them always come to my help, whenever and wherever I stumbled. Special thanks to Mitsu Mam, without her I can't even imagine to be at this stage.

I would also like to thank all the members of CADLAB who were very enthusiastic in solving and discussing research problems during Lab talks. I have already started missing those talks.

I would like to thank all the professors from SERC and CSA, for the great courses that they offered. It was a great learning experience sitting through each and every one of the lectures. I would also like to thank SERC and IISc for giving me opportunity to be part of this wonderful campus and ambiance to work in. Special thanks to Mallika mam for taking great care of every student in the department and also for forwarding the scholarship request on time.

I would like to thank all my SERC friends, Maharashtra Mandal Guys, my Hostel Friends, my Seniors, my Juniors for making this two year long journey sweet and memorable. I will cherish every moment spent with you guys.

I am thankful to everyone who were beside me during the course . It would have been impossible to complete my project without the support and invaluable help of the kind people around me.

Finally, Mom and Dad, I am eternally indebted for your love and affection. There is no word to express my gratitude toward you.

Abstract

Virtualization has emerged as an answer to many critical issues being faced by enterprise computing infrastructure. These includes workload consolidation, Quality of Service QOS and much more. Many of these systems are implemented through a software abstraction layer also knows as hypervisor. Unfortunately, the use of virtualization is the source of a significant security and performance concern. Multiple virtual machines run on the same hardware and since the virtualization layer plays a considerable role in the operation of a virtual machine, a malicious party has the opportunity to attack the hypervisor. Due to the mutual interference between VMs there can be significant drop in the performance of a individual VM.

The virtualization technologies have evolved on systems that were not designed for virtualization and hence software layers handle the virtualization of system resources. As a result almost all the techniques are built on time-multiplexing of resources to handle virtualization. However, this timesharing of resources, especially memory and I/O devices with the hypervisor being the arbitrator for most virtual machine (VM) actions on allocated memory and I/O devices has resulted in hypervisors with complex design and functionality and mostly serialized access to I/O devices. Also, virtualized environments in current technology provide isolation between VMs that is limited to the software stack residing inside the VM. The system hardware and the hypervisor controlling this hardware are still shared amongst these VMs. As a result, the hypervisor becomes a single point of failure on the virtualized server. The hypervisor being more complex provides a larger attack surface and is difficult to evaluate using formal methods. So it becomes difficult to test the hypervisor for efficacy in trust-based systems.

We propose a trend shift of building virtualization constructs from time-sharing of resources to space-sharing at the system hardware level. The hypervisor will be limited to managing the hardware resources and its main function is to carve out a subset of physical resources, called system tile and bind that system tile to the requesting VM. A system tile will be a essentially a spatial partition of the complete system. The key idea of spatial partition is to establish isolated regions in hardware and associate access control to a designated VM. All accesses to this system tile are limited to and from the VM associated with it. This eliminates the hypervisors role in

resource access. As a result the virtualization overheads can be reduced since the VM can now directly access the system tile resources which are not shared with any other VMs.

For evaluating the proposed architecture, we have performed different experiments which covers I/O, memory and CPU intensive workloads. These experiments are performed on virtualized as well as non-virtualized setup on the existing virtualization technologies. We observe the performance gap between the virtualized setup and non-virtualized, attributed to software layer used for providing the virtualization. This performance gap widens as we increase the number of VMs. Thereafter an intuitive study is done to show how the proposed architecture eliminates the performance gap, thus making virtual machine as efficient as non-virtualized machine. In this work we will show the carving of system tile from the pool of underlying resources. We will also see how security issues for accessing various resources can be handled with simple and elegant construct that are built on hardware APIs.

Contents

List of Figures

List of Tables

Chapter 1

Virtualization, Pitfalls and toward Architecture Proposal

In this chapter, we will give a brief introduction to virtualization technology. Then we will discuss some performance and security related concerns with the current virtualization technologies. Then we give a brief introduction to proposed architecture.

1.1 Introduction

Emerging trend of multi-core servers promises to be the solution for all cloud infrastructure related issues with system virtualization as the enabling technology. System virtualization allows to create virtual replicas of the physical system, over which independent virtual machines can be created, complete with their own, individual operating systems, software and applications. This provides total system isolation of the virtual machines. Apart from this, the key driver for virtualization adoption in systems will be virtual machine performance and security isolation that can be achieved over a consolidated server with shared resources.

Most contemporary virtualization technologies have evolved on system architecture built for single system use. As such, these technologies depend on the time-sharing model for resource virtualization. Since the system hardware is not completely designed for virtualization, the idea of virtual resources is a software abstraction that is built within the virtualization layer. This virtualization layer becomes the single point of failure for all the VMs sharing a specific resource. For example, an I/O device like the network interface card is virtualized using either para-virtualized split driver model (Xen) or through emulation using Qemu (VMWare and KVM) [7, 8, 14]. In either of these virtualization techniques, the native device driver is resident within a protected OS environment, which is a general purpose OS and if this is compromised,

all VMs associated with the device or the host are compromised. Also, since the resource usage controls are defined on the virtual devices manifested in the software virtualization layer, the physical resource can be overwhelmed by a malicious virtual machine posing to share a common resource like the NIC. This can cause a denial of service (DOS) like attack on the system and virtually render all VMs on it to be inaccessible. The hypervisor layer is also the source of a major security concern as malicious parties can get direct access to the server where their victim may be running. The malicious party can actively attack the virtualization layer. If successful, many vulnerabilities have shown to be possible [22, 15], the attacker has an elevated level of execution capabilities on a system running other virtual machines. The malicious party can then inspect the memory, exposing confidential information such as encryption keys and customer data, or even modify the software a virtual machine is running. Even without compromising the hypervisor, hosting multiple virtual machines on a single physical machine expose side-channels that can be used to learn confidential information [17]. These security risks may make customers hesitant to use hosted virtualized infrastructures.

As said earlier, prevalent virtualization technologies have evolved on systems that were not designed for virtualization and hence software layers handle the virtualization of system resources. As a result almost all the techniques are built on time-multiplexing of resources to handle virtualization. However, this time-sharing of resources, especially memory and I/O devices with the hypervisor being the arbitrator for most virtual machine (VM) actions on allocated memory and I/O devices has resulted in hypervisors with complex design and functionality and mostly serialized access to I/O devices. Memory virtualization has made progress from software controlled shadow pages to hardware solutions such as Intel's Extended Page Tables (EPT) or AMD's Nested Page Tables (NPT)[16] but there is still a significant overhead in second level address translation. The SIG-PCIes SR-IOV [10] specifications provide for some virtualization enhancements on the I/O devices, but the way hypervisors expose the virtual device interfaces and enables access to them still carry some overheads essentially because of time-sharing constructs. This causes an impact on application/VM performance since the memory and I/O resources allocated to the VM are not always under the direct control of the guest OS running on the VM. Overall, the time sharing virtualization models lead to performance and security overheads on virtualized servers leading to reduced device bandwidths for the VMs and limited scalability for sharing the I/O device.

Virtualized environments in current context provide isolation between VMs that is limited to the software stack residing inside the VM. The system hardware and the hypervisor controlling this hardware are still shared among these VMs. As a result, the hypervisor becomes a single point of failure on the virtualized server. For example, there is a single point of failure for

all VMs sharing a device due to native device driver being hosted in the VMM or hypervisor Domain. There is also potential vulnerability for denial of service attacks due to performance interference. Data privacy issues have to be considered where third party cloud providers are involved. The hypervisor being more complex provides a larger attack surface and is difficult to evaluate using formal methods. So it becomes difficult to test the hypervisor for efficacy in trust based systems.

1.2 A Brief Introduction to Proposed Architecture

Rather than attempting to make the virtualization layer more secure by reducing its size, we instead take the position that the virtualization layer should be removed altogether. In this work we propose getting rid of the virtualization layer running beneath each guest operating system (OS). As a side benefit, removing the active hypervisor removes the virtualization tax which is incurred when needing to invoke a hypervisor for many operations.

We propose a trend shift of building virtualization constructs from time-sharing of resources to space-sharing of resources at the system hardware level. The hypervisor now is limited to managing the hardware resources and its main function is to carve out a subset of physical resources, called system tile and bind that system tile to the requesting VM. A system tile is essentially a spatial partition of the complete system, established using re-configurability in hardware. The key idea of spatial partition is to establish isolated regions in hardware and associate access control to a designated VM. All accesses to this system tile are limited to and from the VM associated with it. This eliminates the hypervisors role in resource access. As a result the virtualization overheads can be reduced since the operating system residing in the VM can now directly access the system tile resources which are not shared with any other VMs.

Secondly, the hypervisor's role is now restricted to managing and maintaining the system tile information and VM association to a system tile. Since the system tile is a spatial partition of the system hardware, each partition can be defined based on the device id or physical address range and access to the resource can be easily enforced by checking for device id association and address range violations. This makes the hypervisor design simpler and reduces its attack surface. This simplicity enables proven, formal methods based evaluation of the hypervisors and facilitates building trusted platforms. Also, if a VM fails or acts maliciously, the failure or attack is limited to that VM.

The important features of our architecture are:

- *Dedicated core* - Each processor core is dedicated to a single VM. Once the core is allocated to a VM it is not allocated to another VM. This prevents interference between different

VMs, mitigates side-channels which exist with shared resources.

- *Memory Partitioning* - Hardware-enforced partitioning of physical memory ensures that each VM can only access the assigned physical memory and only in a fair manner.
- *Dedicated Virtual I/O Devices* - I/O device modifications to support virtualization enables each VM to be given direct access to a dedicated (virtual) I/O device. The memory management facilities along with chip set support ensure that only the authorized VM can access the memory-mapped I/O and only at a given rate.

1.3 Outline of the thesis

Rest of the thesis is organized as follows. Chapter 2 highlights related work to this thesis. In chapter 3 we discuss the various functions of VMM or hypervisor. We move on to architecture proposal in chapter 4. In chapter 5 we describe life-cycle walk-through of a VM on proposed architecture. To highlight virtualization overheads in current technology, we performed different experiments. The results of this experiments are discussed in chapter 6. In chapter 7 we gave glimpse of current work and chapter 8 provides conclusion and future work.

Chapter 2

Related Work

Virtual machine monitors and hypervisors have been studied extensively. KVM, VMware vSphere, xen and microsoft's Hyper-V[21, 7, 14, 13] are some example of software based hypervisor. Currently all of this techniques used the hardware assisted virtualization. Intel VT-x and AMD-V [20] are example of hardware assisted virtualization technology at processor level. Some noteworthy example of hardware enhancement for virtualization by these technology are VMX root operation and VMX non-root operation, the virtual-machine control structure (VMCS) is a new data structure that manages VM entries and VM exits and processor behavior in VMX non-root operations, VPID - virtual processor id indexed caches, new instruction to support virtualization and extended page table.

Chen et. al. [9] proposed a hardware assisted full virtualization technique Dynamic Binary Translation in DIMM (DBTIM). DBTIM translates binary codes between different ISAs and translate the privileged instructions with an equivalent block of unprivileged instructions. This provides hardware full virtualization to both x86 architecture and non-x86 architecture.

Most of the work is related to securing the virtualization layer in a server or cloud setting. This includes work by Santos, et al., [19] and sHype [18] from IBM. Santos, et al., presented a solution which uses the Terra hypervisor [12]. The sHype is a secure hypervisor architecture whose goal is to provide a secure foundation for server platforms. Instead of securing the virtualization layer, we are removing it and absorbing its functionality in the hardware layer.

Chapter 3

Functions of the Hypervisor

We are proposing an architecture which removes the virtualization layer. To better understand the implications of this proposal, we need to first understand the role the virtualization layer plays in today's technology. Below, we discuss the many functions of today's virtualization layers (as used in cloud infrastructures).

3.1 Scheduling Virtual Machines

Since in today's typical virtualized environment, multiple virtual machines are running on a single processor, the hypervisor needs to arbitrate the access to the processor cycles. Much as an OS controls the CPU allocation of running processes, the hypervisor controls the CPU allocation of running virtual machines. Whenever a timer expires, I/O is performed, or a VM exit occurs, hypervisor's scheduler routine is run to decide which VM to run next.

3.2 Memory Management

The hypervisor takes care of managing the physical memory available on the system. Memory is a limited resource which the hypervisor needs to arbitrate and share among the guest VMs. To help with the illusion that each guest VM has its own physical memory, the hypervisor presents each guest VM with its guest physical memory. The hypervisor then maps the guest physical memory to the host physical memory which is the actual physical memory available. Through this remapping of memory, the hypervisor is able to achieve isolation between the guest VMs. Each VM thinks that it has some physical memory, the guest physical memory, and is only able to access that memory. This prevents VMs from accessing the memory of other VMs. current day hypervisor technology use the nested page table (AMD) or extended page table(Intel) to accelerate remapping from guest physical to host physical memory[23].

3.3 Emulating and Arbitrating Access to I/O Devices

Access to the physical devices is essential as the I/O of the system is how a program interacts with the real world. As each device is shared, the virtualization layer must arbitrate accesses to each device. Additionally, the virtualization layer can present a generic view of the underlying device rather than the actual device in order to enable a virtual machine to run on servers with equivalent but different devices. In this scenario, the root context virtual machine runs the real device drivers and emulates the generic device the VMs access.

3.4 Starting and Stopping Virtual Machines

The hypervisor emulates the underlying hardware, giving each virtual machine the view that it is running on its own machine. Management software in the root context can interact with the hypervisor to create a new virtual machine and control the power to the virtual machine (e.g., power on or reset). The hypervisor emulates all of the aspects of the server, enabling the OS to run through its boot sequence.

Chapter 4

Proposed Architecture

Proposed architecture tries to eliminate the resource virtualization layer by restricting the hypervisor for resource management. The resource access mechanism that current hypervisors provide, either using emulation or para-virtualization, is eliminated. This is attempted by system architecture design that allows execution of VMs directly on the hardware without the underlying hypervisor layer participating in resource access. The hypervisor here is a resource management entity whose role is essentially to allocate and de-allocate hardware system resources as specified by a VM and ensure resource isolation across VMs. Due to this reduced role of the hypervisor, this entity is called as the System Configurator. It is envisaged that the System Configurator is the resource control software that is an extension of the BIOS or firmware functionalities of a modern computer. The extensions of the System Configurator are to enable creation of independent system partitions that can be allocated to a specific VM. The VM then boots over this partition, as it would on a native system, and manage the resources allocated to it as its physical resources. The System Configurator now ensures that any resource access from this VM is restricted to the resources of the system partition allocated to it. Hence the System Configurator is built to enable identification and flagging-off of all resource access violations. The system design is enhanced to enable creation of dynamic system partitions and necessary control structures that can provision for identification of resource access violations. The goal of this design is thus to extend isolation of the VM even with respect to the system resources. The key idea behind this design is to exploit spatial partitioning of system resources using reconfigurable architecture design. In prevalent virtualization techniques, where hypervisor plays an important role in resource access by the VM, the proposed architecture deviates in eliminating the role of hypervisor in resource access. Thus in the proposed architecture, once the VM is instantiated it manages access to the resources in its system partition. The hypervisor observes any resource access violations and generates an exception in case of violation.

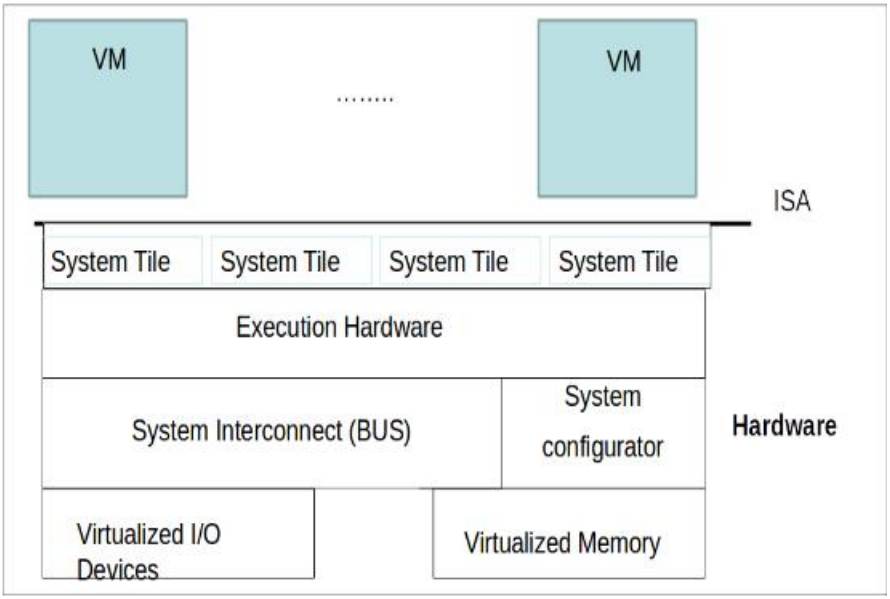

Figure 4.1: Overall System Architecture

4.1 Overall System Architecture

The proposed architecture targets the important issue of performance and protection with micro-architectural support from next generation system on chip (SoC) design. Fig. 4.1 shows the overall system architecture. It consists of execution hardware, virtualized I/O devices and virtualized memory. The hardware devices are connected to each other via system interconnect. The execution hardware consists of large number of CPU cores. Virtualized I/O devices are physical I/O devices extended with capability to spatially partition the device into many device contexts. Virtualized memory is also memory extended with features for spatial partitions into multiple contiguous partitions called memory contexts. Using these features of exclusive physical CPU cores, memory contexts and device contexts a system tile can be constructed. The system tile is thus a collection of complete system resources. With necessary architectural support, the system tile can be defined as a secure subset of hardware resources. We perceive that this definition can be built by extending the system ISA.

4.2 System Tile

System tile is defined as a logical partition of the complete system hardware. Figure 4.2 shows the schematic diagram of a System tile. A full-fledged operating system (OS) can be hosted on the system tile. A system tile is composed of processing cores, memory contexts and I/O device

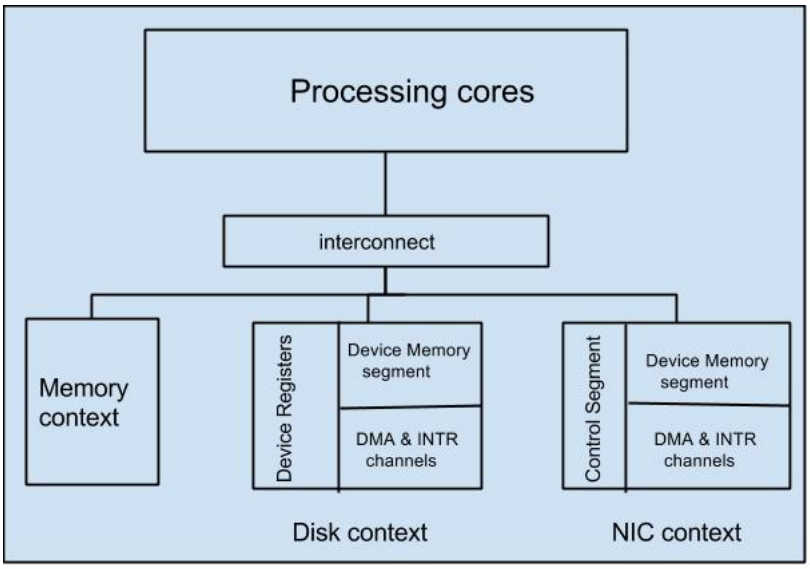

Figure 4.2: System Tile

contexts. The processing cores are the actual physical CPU cores. Memory context is range of physical memory that the VM can access. VMs access of memory beyond this allocated range is treated by the System Configurator as access violation. I/O device context describes the portion of I/O device that is accessible or allocated to VM.

4.3 System Configurator

The System Configurator maintains resource tables that give an idea of the total system resources, total allocated resources to system tiles and which tile is associated with which VM. The System Configurator understands the concept of a VM and maintains its identification by a VM-ID. Each system is at best associated with one VM-ID, to begin with. It is envisioned that the System Configurator will have a programmable interface by way of which an external entity like the cloud resource manager can issue instructions to it for gathering system status and usage information, send out commands for creating, instantiating and shutting down VMs based on their resource demands and lifetimes. As the architecture is proposed have re-configurable capabilities, the System Configurator creates a system tile only when a VM request comes. Once a system tile is created, binding of the tile is achieved while instantiating the VM and then booting of the OS image is initiated.

The System Configurator with this limited functionality can be viewed as a simple finite state machine. As such, the System Configurator can be then be processed through formally

provable methods for security and reliability and thus enabled for building trust-able systems.

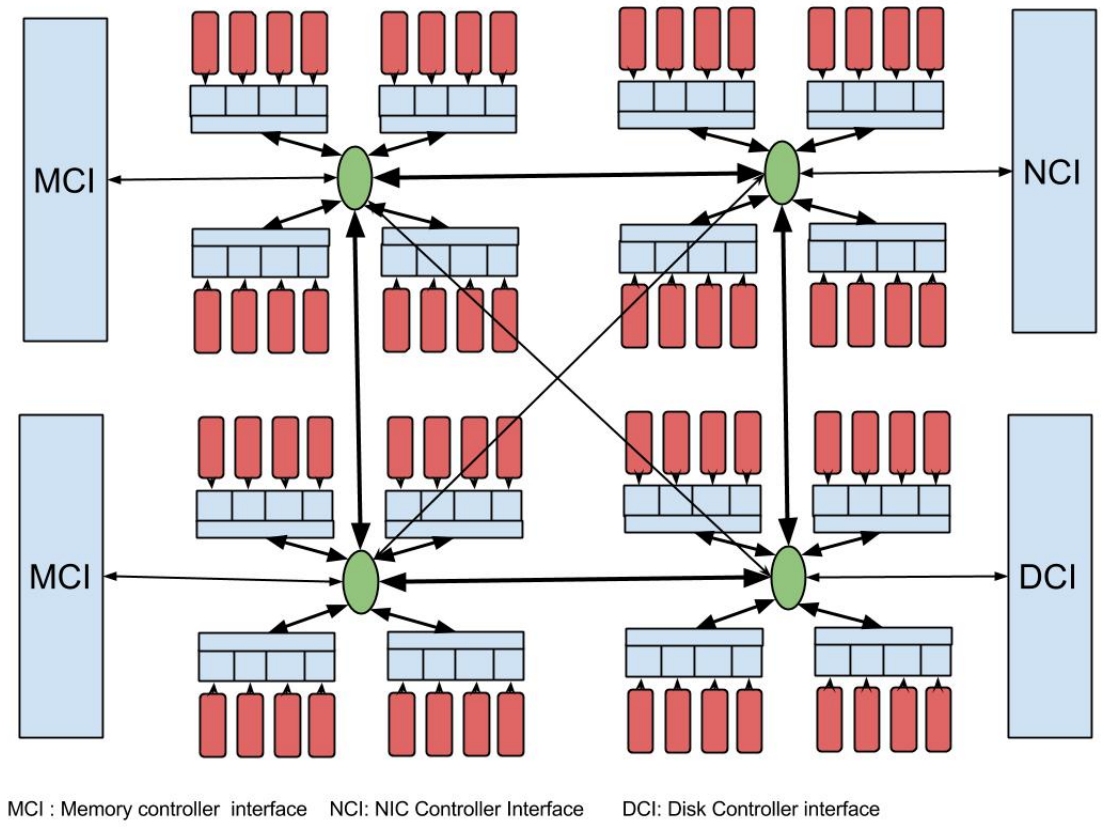

Figure 4.3: System Fabric

4.4 System Fabric

Figure 4.3 shows the conceptual architecture of the system fabric. The system fabric consists of CPU cores, communication routers and different controller interfaces like MCI (Memory Controller Interface), DCI (Disk Controller Interface) and NIC (Network Interface Card) controller interface. The red rectangle represents the CPU cores. These CPU cores include the mechanisms needed within the core to support a reliable operating system. The blue boxes just below the CPU cores are the last level caches. These caches are associated with the corresponding CPU core only. If a cache miss happens at this cache level then the request for particular data is injected into the on-chip NoC. The blue horizontal bar below the last level cache represents the entry point to on-chip NoC. The CPU cores on the chip are connected to other cores and also other system resources via the interconnection routers. The green ovals represent the on-chip

routers. All the systems resources except the CPU cores are defined as contexts to compose the system tile. These contexts are essentially identified by the specific device address range. The router routes the data packets to appropriate location based upon the destination address of the request.

4.5 System Resource division

Since the design goal of the architecture is system isolation to leverage performance and security benefits, the allocation constructs for the resources are such that no two VMs have any common resource shared among themselves. This is achieved by making physical CPU core allocation to a VM exclusive. Every device context is also exclusive and is not shared among VMs. As stated earlier, resources like memory and I/O devices are exposed as device contexts by the virtualization enabled memory and I/O devices. Each device context is basically a small spatial partition of the basic resource.

4.5.1 Memory Context

Memory context is a segment of physical memory that the VM is permitted to access. The System Configurator allocates the memory segment to a VM based on its requirement and the memory context is defined by the address range for this segment. The memory context is then visible to the VM as its own RAM which can be accessed by initializing a base address register with the starting address of the memory segment allocated to the VM. The size of the memory segment is passed to the VM as its RAM size during boot-up. The System Configurator stores complete information of each memory segment, its address range and the association with a VM. To be able to use this mode of segment allocation, it is envisaged that the VMs OS must have a re-locatable kernel. A re-locatable kernel can boot from any RAM address. This kernel uses the starting address of the memory context as its base address and the system instantiates the booting of the re-locatable kernel in that memory context. The use of memory context and a re-locatable OS kernel eliminates the need for second level page address translation found in prevalent virtualization architectures. Hence, it is expected to improve the performance of memory accesses in the VM. Secondly, each memory context is bound based on the address range, access violations are easily detected based on the assigned memory bounds.

4.5.2 Device Contexts

This system design has a generic principle of building virtualization enabled devices that can support multiple spatial partitions of the device, which can then be exposed and assigned to a system tile as a device context. The spatial partition concept is identified basically by the

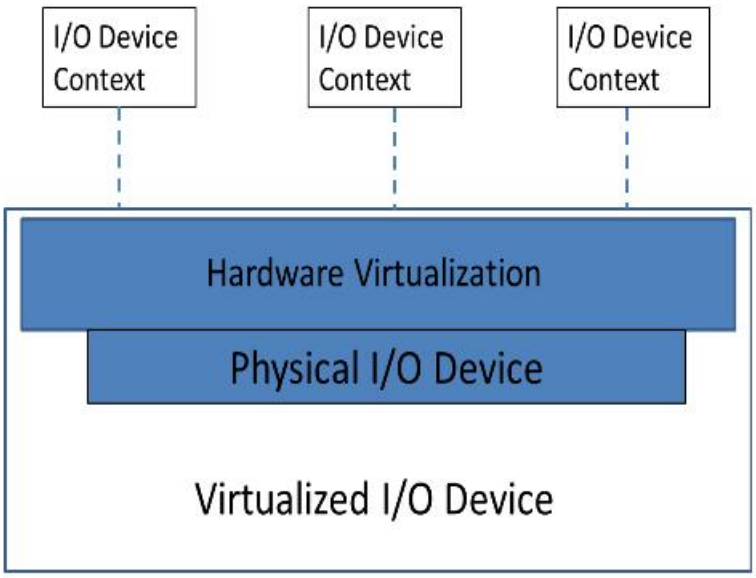

Figure 4.4: Device Virtualization

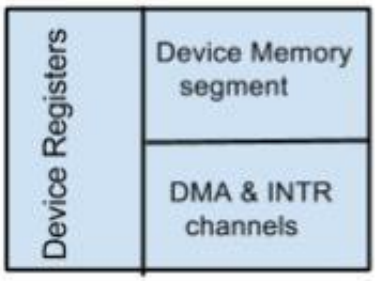

Figure 4.5: Concept of a Device Context

device address range associated with a device context. Given that most I/O devices in today's context support memory mapping, the access range and its violation is then simplified to the task of accessing the specified address ranges associated with a given system tile. This enables the building of simple constructs for checking violations. Figure 4.4 gives a schematic diagram a physical device supporting multiple device contexts. The hardware virtualization layer in the device provides the ability to spatially partition the physical device into different device contexts based on the device requirements from a VM. Each device context is allocated exclusively to a VM and is exported as a physical device to the VMs OS kernel during boot-up. The device context also ensures the resource usage limits associated with its sizing and thus gives fine

grained control on the device use by any VM. As a result this provides inherent architecture support for preventing DOS kind of attacks through I/O requests.

Figure 4.5 depicts the concept of a generic device context. The various components of a device context are explained below:

1. *Device registers*: A dedicated set of registers is used by each device context to indicate the status of device context associated operations. These are akin to the normal I/O device registers used for configuration, control and transfer related operations.

2. *DMA and INTR channel*: Each device context is provided with a dedicated DMA channel to enable data transfers from and to the device memory and the address space of VM. To initiate or acknowledge data transfer the device context is also associated with an independent interrupt line by way of message signaled interrupts (MSI).

3. *Device memory segment*: Each device context is associated with a partition of device memory. The System Configurator maps this memory segment into the I/O address space of the VM. It is envisaged that I/O device contexts are available using memory mapping of the I/O devices. The size of the device memory partition is arrived based on the bandwidth requirement for the device context.

Chapter 5

Walk-through of VM life cycle

5.1 VM creation

When a VM creation request arrives from an external entity(e.g. Cloud manager) , the System Configurator checks for the available resources, from the resource tables that it maintains and if the resources are available, System Configurator creates a system tile for those resources marks them as allocated. The System Configurator associates the system tile with a VM-id and sends back the acknowledgement to the request from the cloud resource manager with a VM-id and system-tile id. The cloud manager then can initiate a VM boot onto the system tile with the given VM-id.

5.2 Possible access violation from VM

When a System Configurator gets a request for VM creation, it establishes the boundaries of various resources like memory, device contexts and CPU cores. These system resource boundaries are conveyed to the VM by the system tile id. The system tile defines the physical limits of the associated resources by a VM. Any faults that arise in VM by the applications running inside it can be handled by the VM itself. Like an application seeking memory that is outside its process address space is handled exactly the way it is handled in any non-virtualized environment. However, when a VM tries to access the resources outside its allocated range as in case of buggy or malicious software, the underlying hardware will produce faults and raise an exception for the VM or the cloud manager. The VM or the cloud manager can then take appropriate action depending on the nature of exception. This can be easily implemented by keeping base and bound registers for memory and device contexts at each system tile and monitoring each memory or device access request.

5.3 VM shutdown and system tile revocation

The situation for system tile revocation may arise for two reasons. First, a VM may voluntarily release the resources allocated to it when it initiates a shutdown. The second case is when a VM is accessing memory or device context outside its allocation area and the cloud manager decides to shutdown the VM. In the first case, this architecture requires some support from the OS running on the system tile. When system is shutdown it needs to signal the System Configurator about its status so that System Configurator is able to free resources allocated for that system tile. In the other case, when the hardware raises an exception for access violation, the System Configurator must be able to send this information to the cloud manager. Based on the nature of exception, the cloud manager can decide to issue a shutdown message to the VM which then will go as per the procedure described for the earlier case.

Chapter 6

Experiments and Results

We performed some experiments to demonstrate the interference in the VM performance in virtualized environment with Memory, Disk and CPU workloads. The purpose of these experiments is to understand the effect of software based hypervisor solutions on applications because of system resource sharing. Current solutions achieve resource sharing mostly by time-sharing resources across multiple VMs. Each case study exposes the effect of one of the system resource, namely CPU, Memory and Disk space. We draw the analogy of VM provisioning in our proposed architecture to the case of non-virtualized result. This is intuitive since we adopt space-sharing as the principle behind resource sharing in our solution.

6.1 Experiment Setup

The experiments are conducted on a virtualized environment with KVM as the hypervisor to set up VMs. The host machine has quad-core intel core i7 Processor, 8GB RAM and 1TB SATA2 disk(7200 rpm,16MB cache). A dedicated partition on the disk is used for all VMs to minimize interference of host operating system. All the VMs are configured identically with 30GB virtual disk space, 1GB RAM and 1 vCPU pinned to an exclusive core so as to avoid any CPU scheduling effects. Also their non-VM counterpart is booted with 1 physical CPU and RAM restricted to 1GB. To model the I/O patterns of real world applications, the *IOzone* file system benchmark tool [1] is used. The I/O operations are configured to eliminate cache and buffer effects so as to get native read/write performance for the disk. In case of CPU we have used *nbench* bench-marking tool[2] is used and for measuring the memory performance we have used *STREAM* benchmarks [4] and linux perf tool [3] to measure last level cache behavior. We have run all of the mentioned benchmark on multiple VMs simultaneously. Enough care is taken to run the benchmark on each VM for same amount time.

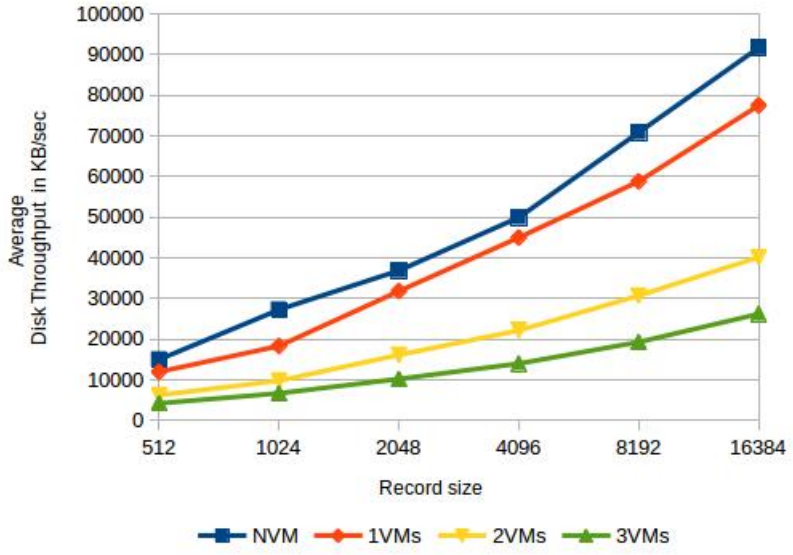

Figure 6.1: Average disk throughput with varying record sizes for different number of VMs

6.2 Effect of VM interference on Disk Performance

We examine the variation in average disk throughput with different number of concurrent VMs against varying record sizes for write operations. I/O record sizes affects the net throughput achievable as indicated in Figure 6.1. It is observed that there is a considerable drop in disk throughput (from 91 MB/s to 77 MB/s) in a virtualized environment as compared to the non-virtualized case where the I/O operation is executed directly on the host. This indicates that the hypervisor causes an overhead for the I/O applications running inside VMs. It is also observed that when the number of VM is increased, the virtualized disk throughput (77 MB/s in this case) is equally divided into currently running VMs on a physical machines due to default fair scheduling. Thus there is no resource usage limit is enforced by the hypervisor.

In virtualized environments, all I/O requests from the VMs are dispatched to the actual disk by the disk scheduler of the host. When I/O request is generated by an application inside the VM, To handle this, a trap is generated and there is a switch to the kernel mode from the guest mode. It is examined if the reason of the switch is an impending I/O operation in which case the control is passed to the corresponding qemu-kvm process in user space to perform I/O on behalf of the VM. Then the I/O request is completed in the same way using the system calls as done for other normal processes running within the Linux operating system. Due to the mode switches associated with every I/O request, there is an inherent virtualization

Operations	NVM	1VM	2VM	3VM
Num. sort	2076	2068.2	1989	1988.067
Str. sort	874.89	333.04	318.525	319.330
Bit	6.47E+008	6.44E+008	6.16E+008	6.17E+008
FP Emu.	334.31	333.03	318.635	319.193
Fourier	40358	40181	38503	38540.667
Assign	48.82	48.619	46.601	46.658
Idea	8756.8	8706.6	8354.45	8358.267
Huffmann	3071.6	3061.1	2935.1	2940.133
Neural	94.614	94.298	90.0265	90.085
LU Deco.	2288.5	2279.8	2182.15	2190.167

Table 6.1: Average CPU performance for different number of VMs

overhead associated with the I/O operations executed from inside a VM and this overhead is a reason for decrease in the bandwidth of the disk in virtualized environment as compared to non-virtualized. In the proposed architecture it is expected that there will be no virtulization overhead as each virtual machine is given a direct control over the device context of virtualized device.

Moreover, we have seen that when the number of VMs are increased, the bandwidth for virtualized environment is divided equally. This could cause a DOS where the multiple malicious VM can increase their disk usage to lower the disk bandwidth of their victims. Thus in this scenario a restricted device usage would be more favored. Disk bandwidth usage restriction can be enforced on each VM by the virtue of device memory buffer allocated to disk device context.

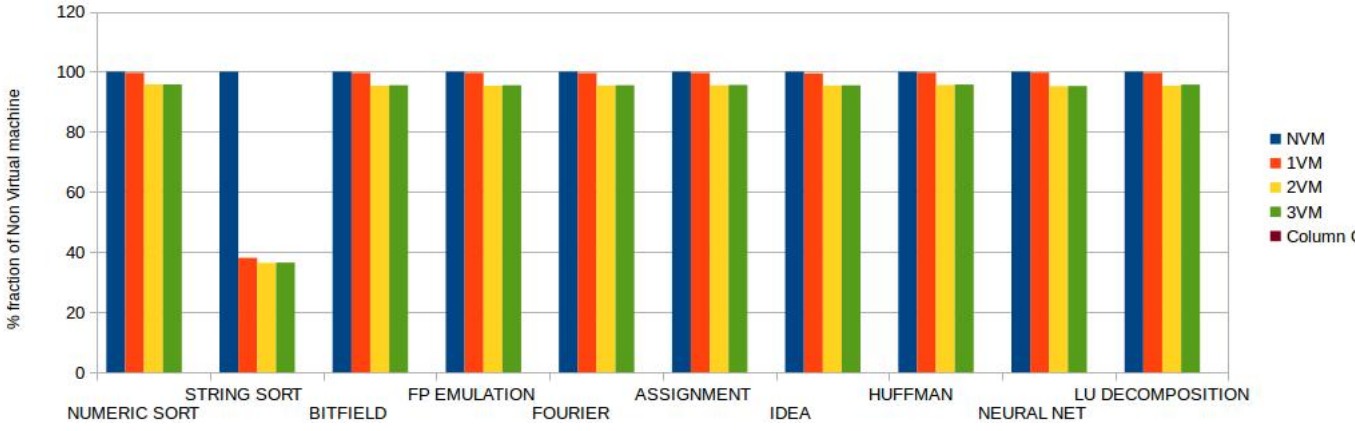

Figure 6.2: Average CPU performance for different number of VMs

6.3 Effect of VM interference on CPU Performance

In case of CPU workload, we compare the average CPU performance of native machine to virtual machine using *nbench*. The performance is reported in terms of iteration per second. for example in Numeric sort this benchmark performs numeric sort iteration for some finite time and reports the iterations executed in that amount of time. Figure. 6.2 shows the normalized benchmark results with respect to non-virtualized VM. In Table 6.1 and in figure 6.2 we observe that there is no performance difference between virtualized and non-virtualized environment. Significant performance difference is observed only in case of String sort. In this case native machine perform approximately 3 times better than its virtualized counterpart. In String sort the comparison is done between 2 strings which does not follow spatial locality principle. For each cache miss there is page table walk of 8 levels in virtualized environment and in non-virtualized environment page table walk is limited to only 4 levels [5]. Thus, in case of string sort we observe a significant performance difference and all operations other than String sort follows the principle of spatial locality. From this experiment we conclude that, in the case of CPU there are no significant virtualization overheads except in some cases where spatial locality of data is not satisfied.

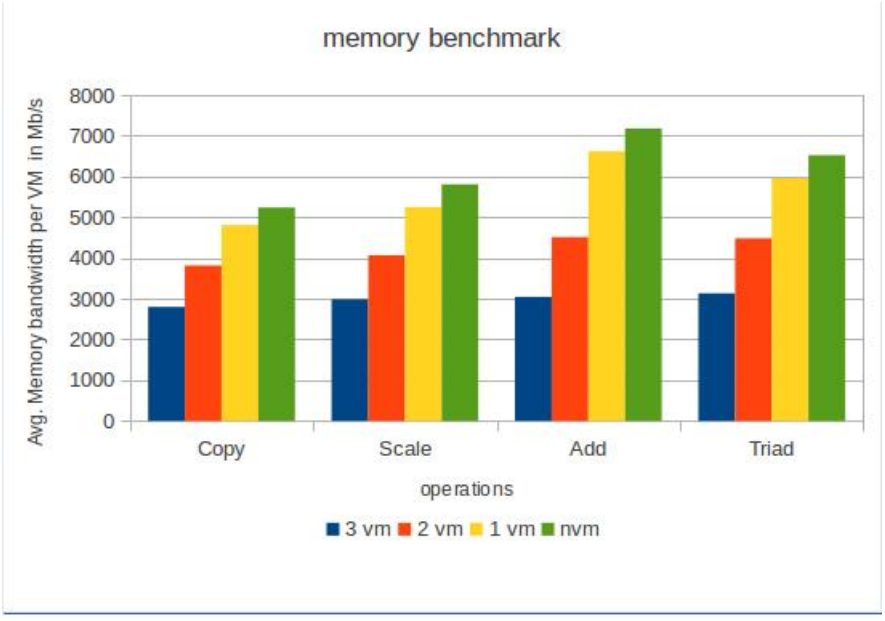

Figure 6.3: Average Memory bandwidth for different number of VMs

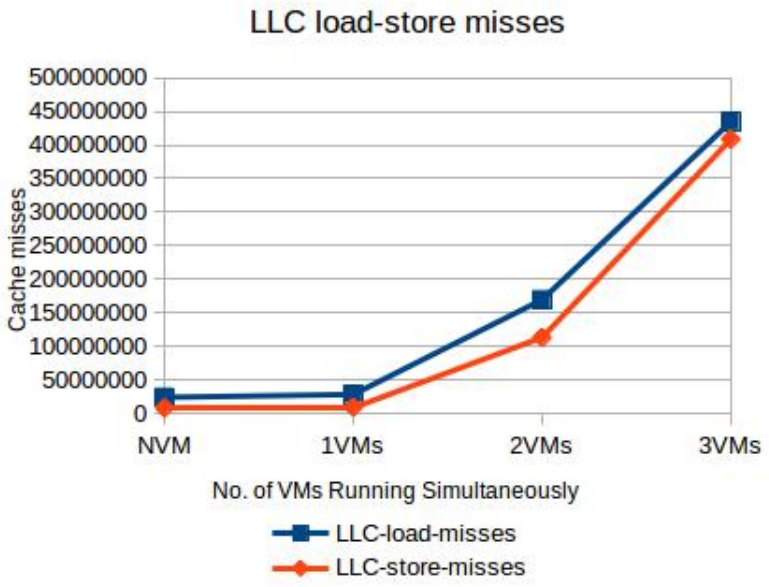

Figure 6.4: Last Level Cache load-store misses for different number of VMs

6.4 Effect of VM interference on Memory Performance

We examine the variation in average Memory Bandwidth achievable with different number of concurrent VM.

In Figure 6.3 we observe that the memory bandwidth offered to virtualized machine is lower than memory bandwidth offered to non-virtualized machine. In *STREAM* benchmark the bandwidth is measured from inside the VM. In order to bring the same amount of data from the memory the virtualized setup has to spend more effort to bring data from memory than the non-virtual machine because of second layer address translation. As we increase the number of VMs the average Memory bandwidth decreases. It may seem that the cumulative bandwidth of 2 VMs or 3 VMs is greater than native machine. This can be due multiple hardware enhancement like hardware pre-fetching and cache level hierarchy.

To get the clear idea of actual memory transfer from RAM to processor and other way, we measured the last level cache behavior for different number of VMs. When there is miss for load or store instruction, an actual memory request is generated to transfer the data to and fro from processor cache hierarchy and memory.

In Figure 6.4, we observe that there is 17 and 13 percent increase in last level load and store misses respectively. This increment in last level cache misses is due to the virtualization

overhead of second layer address translation which is responsible for the decrease in bandwidth in virtualized environment. When we increase number of VM to 2 and measured the last level of cache misses we observe that for LLC load and store about 8 and 10 fold increase respectively over single VM. And when concurrent number of VMs are increased to 3 the LLC load-store cache misses increases 2.5 and 4 fold increase over 2 VM case. This LLC misses increases due to common cache shared between processor core and hence with different VMs. This increase in cache misses results into the increases memory bandwidth utilization.

In case of our proposed architecture we do eliminate the need of the second level address translation and we plan use a contiguous block of memory instead of segmented. So the virtualization overheads of second layer address translation are not present in this architecture. It also provide more than one on-chip memory controller to minimize the memory bandwidth contention by competing VMs. A placement policies VM can be developed to use memory bandwidth offered by the memory modules optimally.

Chapter 7

Current Work

By the time of writing this theses we have developed a NoC which based on OpenSoC Fabric open source project [11]. This NoC uses wormhole routing and an Dimension order routing function based on the addresses of the endpoints to which physical CPU cores will be attached and the address of the memory location. We are planning to use Xilinx microblaze IP attached to the endpoints of the NoC on which a lightweight linux kernel can be booted. The NoC is written in high level hardware description language *Chisel* [6]

Chapter 8

Conclusion and Future Work

One significant source of the problem is the use of software virtualization, where an increasingly complex hypervisor is involved in many aspects of running the guest virtual machines. Rather than focus on making virtualization more secure, we remove the need for an active virtualization layer in the first place. We presented an architecture which promises benefits on par with those of todays virtualization solutions without the active virtualization layer. Instead, our architecture establish isolated regions in hardware and associate access control to a designated VM. All accesses to this system tile are limited to and from the VM associated with it. Hypervisor's or System Configurator's role is now restricted to managing and maintaining the system resource information and VM association to a bunch of system resources also known system tile. Key aspects of the architecture include:

1. Dedicated cores for VM.
2. Hardware enforced memory partitioning.
3. Dedicated virtual devices that is controlled by each guest VM directly.

In this work, we have presented a virtualization aware architecture. Toward the realization of this architecture a lot has to be done. Following are some points we need to be considered for future work. First of all we need to develop a cache coherence protocol on top of the NoC which will allow sharing of the last level cache between different cores belonging to same VM. In this architecture System Configurator plays an important role in terms resource allocation. This System Configurator has to be visualized as part of NoC fabric and has a specialized data structure which represents the resource allocation. Thus we need to design System Configurator as a specialized core which will use packed based command on NoC to manage the resource allocation to VMs. On the endpoints of the NoC we plan to connect Xilinx Microblaze IP as

a soft-core processor. To impose the commands issued by the System Configurator we need to design a wrapper around the processor.

Bibliography

[1] URL http://www.iozone.org/. 17

[2] URL http://www.tux.org/~mayer/linux/bmark.html. 17

[3] URL https://perf.wiki.kernel.org/. 17

[4] URL https://www.cs.virginia.edu/stream/. 17

[5] Amd-v nested paging, 2008. 20

[6] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1216–1225. ACM, 2012. 23

[7] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003. 1, 5

[8] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005. 1

[9] Wei Chen, Hongyi Lu, Li Shen, Zhiying Wang, Nong Xiao, and Dan Chen. A novel hardware assisted full virtualization technique. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pages 1292–1297. IEEE, 2008. 5

[10] Yaozu Dong, Zhao Yu, and Greg Rose. Sr-iov networking in xen: Architecture, design and implementation. In *Workshop on I/O Virtualization*, 2008. 2

[11] Farzad Fatollahi-Fard, David Donofrio, George Michelogiannakis, and John Shalf. Opensoc fabric: On-chip network generator: Using chisel to generate a parameterizable on-chip

interconnect fabric. In *Proceedings of the 2014 International Workshop on Network on Chip Architectures*, pages 45–50. ACM, 2014. 23

[12] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 193–206. ACM, 2003. 5

[13] Forbes Guthrie, Scott Lowe, and Kendrick Coleman. *VMware vSphere design*. John Wiley & Sons, 2013. 5

[14] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007. 1, 5

[15] Kostya Kortchinsky. Hacking 3d (and breaking out of vmware). *BlackHat USA*, 2009. 2

[16] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3), 2006. 2

[17] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009. 2

[18] Reiner Sailer, Enriquillo Valdez, Trent Jaeger, Ronald Perez, Leendert Van Doorn, John Linwood Griffin, Stefan Berger, Reiner Sailer, Enriquillo Valdez, Trent Jaeger, et al. shype: Secure hypervisor approach to trusted virtualized systems. *Techn. Rep. RC23511*, 2005. 5

[19] Nuno Santos, Krishna P Gummadi, and Rodrigo Rodrigues. Towards trusted cloud computing. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 3–3. San Diego, California, 2009. 5

[20] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L Santoni, Fernando CM Martins, Andrew V Anderson, Steven M Bennett, Alain Kagi, Felix H Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005. 5

[21] Anthony Velte and Toby Velte. *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc., 2009. 5

[22] Rafal Wojtczuk. Subverting the xen hypervisor. *Black Hat USA*, 2008, 2008. 2

[23] Sheng Yang. Extending kvm with new intel® virtualization technology. In *Intel Open Source Technology Center, KVM Forum*, 2008. 6