NoHype: Virtualized Cloud Infrastructure without the Virtualization

Eric Keller Jakub Szefer Jennifer Rexford Ruby B. Lee

Princeton University, Princeton, NJ, USA
{ekeller, szefer}@princeton.edu jrex@cs.princeton.edu rblee@princeton.edu

ABSTRACT

Cloud computing is a disruptive trend that is changing the way we use computers. The key underlying technology in cloud infrastructures is virtualization – so much so that many consider virtualization to be one of the key features rather than simply an implementation detail. Unfortunately, the use of virtualization is the source of a significant security concern. Because multiple virtual machines run on the same server 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 virtualization layer. A successful attack would give the malicious party control over the all-powerful virtualization layer, potentially compromising the confidentiality and integrity of the software and data of any virtual machine. In this paper we propose removing the virtualization layer, while retaining the key features enabled by virtualization. Our NoHype architecture, named to indicate the removal of the hypervisor, addresses each of the key roles of the virtualization layer: arbitrating access to CPU, memory, and I/O devices, acting as a network device (e.g., Ethernet switch), and managing the starting and stopping of guest virtual machines. Additionally, we show that our NoHype architecture may indeed be "no hype" since nearly all of the needed features to realize the NoHype architecture are currently available as hardware extensions to processors and I/O devices.

Categories and Subject Descriptors

C.1.0 [**Processor architectures**]: General; D.4.6 [**Operating systems**]: Security and protection—*invasive software*

General Terms

Design, Management, Security

Keywords

Cloud computing, Multi-core, Security, Hypervisor, Virtualization

1. INTRODUCTION

There is no doubt that "cloud computing" has tremendous promise. The end user of a service running "in the cloud" is unaware of how the infrastructure is architected – it just works. The provider of that service (the cloud customer in Fig. 1(a)) is able to dynamically provision infrastructure to meet the currene demand by leasing resources from a hosting company (the cloud provider). The cloud provider can leverage economies of scale to provide dynamic, on-demand, infrastructure at a favorable cost.

While there is debate over the exact definition, the main idea behind cloud computing, common to all approaches, is enabling a virtual machine to run on any server. Since there are many customers and many servers, the management of the infrastructure must be highly automated – a customer can request the creation (or removal) of a virtual machine and without human intervention a virtual machine is started (or stopped) on one of the servers. To take advantage of the economic benefits, the cloud providers use multi-tenancy, where virtual machines from multiple customers share a server.

Unfortunately, this multi-tenancy is the source of a major security concern with cloud computing as it gives malicious parties direct access to the server where their victim may be executing in the cloud. The malicious party can actively attack the virtualization layer. If successful, as many vulnerabilities have shown to be possible [1, 2, 3, 4, 5, 6], 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, multi-tenancy exposes side-channels that can be used to learn confidential information [7]. These security risks make companies hesitant to use hosted virtualized infrastructures [8].

In fact, if not for this security concern, running applications in the cloud can actually be *more* secure than when run in private facilities. Commonly cited are the economic benefits that the economies of scale provide to the cloud infrastructure providers [9]. There is a similar principle with regards to security that is, however, not often discussed. In many organizations, physical security is limited to a locked closet which stores the servers in the company's office. Since cloud providers are served out of large data centers, there are surveillance cameras, extra security personnel, and by the very nature of the environment, the access is much more controlled. That level of physical security is cost prohibitive

ISCA'10, June 19–23, 2010, Saint-Malo, France.
Copyright 2010 ACM 978-1-4503-0053-7/10/06 ...$10.00.

for a single organization, but when spread out across many, it almost comes for free to the customer of the cloud. Similarly, in a private organization, network security of servers is commonly limited to a firewall. Cloud providers can install and maintain special intrusion detection (or prevention) systems which inspect packets for matches to known attacks that exploit bugs in commonly used software. As with physical security, these devices may be cost prohibitive for a single organization but can be provided by the cloud provider for a small cost.

Rather than attempting to make the virtualization layer more secure by reducing its size or protecting it with additional hardware [10], we instead take the position that the virtualization layer should be *removed altogether*. In this paper we propose getting rid of the virtualization layer (the hypervisor) running beneath each guest operating system (OS) in order to make running a virtual machine in the cloud as secure as running it in the customer's private facilities – and possibly even *more* secure. 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 argue that today's virtualization technology is used as a convenience, but is not necessary for what cloud providers really want to achieve. We believe that the key capabilities are the automation, which eases management of the cloud infrastructure by enabling the provisioning of new virtual machines on-the-fly, and multi-tenancy, which allows the cloud provider to gain the financial benefits of sharing a server.

To remove the virtualization layer, we present the NoHype architecture, which addresses, and renders unnecessary, each of the responsibilities of today's hypervisors[1]. This architecture is an entire system solution combining processor technology, I/O technology, and software in order to realize the same capabilities enabled by today's virtualization in a cloud infrastructure, yet done without an active virtualization layer running under a potentially malicious guest operating system. The main components of the architecture revolve around resource isolation. The NoHype architecture automatically dedicates resources to each guest VM, and the guest VM has full control of these resource throughout its runtime on the physical machine. The important features of our architecture are:

- **One VM per Core** - Each processor core is dedicated to a single VM. This prevents interference between different VMs, mitigates side-channels which exist with shared resources (e.g., L1 cache), and simplifies billing, in terms of discrete *compute units*[2]. Yet, multi-tenancy is still possible as there are multiple cores on a chip.

- **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 chipset support ensure that only the authorized VM can access the memory-mapped I/O and only at a given rate.

Removing the active virtualization layer brings significant security benefits. Doing so comes at the cost of not being able to (i) sell in extremely fine grain units (e.g., selling 1/8th of a core) and (ii) highly over-subscribe a physical server (i.e., sell more resources than are available). However, we do not see either of these as a serious limitation. As processors have increasingly many cores on them, the granularity of a single core will become finer with each multicore processor generation (e.g., a single core in a 4-core device is 25% of the system's compute power, in a 128-core device, it's less than 1% of the compute power and it is doubtful applications have such consistent load that expansion and contraction of compute units at a finer granularity than core units would even make sense). Also, attempting to over-subscribe a server is counter to the model of what cloud computing provides. As opposed to private infrastructures where the goal is to maximize utilization of the server by using techniques that adjust the allocation of resources across virtual machines based on current usage [11], in cloud computing the customer is promised the ability to use a certain amount of resources. The customer chooses the amount of resources it needs to match its application. If the customer needs some resources, it pays for them, if not, the cloud provider can assign them to another customer – but does not sell to another customer the resources already promised to a customer. That said, we do not require that the entire cloud infrastructure use the NoHype architecture – a provider may offer a discount to customers willing to forgo either the security benefits or the resource guarantees, enabling the provider to over-subscribe a sub-set of servers.

The NoHype name comes from the radical idea of removing the hypervisor. It also implies that it is not hype, but indeed implementable. In this paper, we also discuss currently available hardware and show that nearly all of what is needed to realize the NoHype architecture is available from computer hardware vendors. As such, we are proposing a commercially viable solution. This does not mean that the current architectures are ideal – we also wish to stimulate the architecture community to think along the lines of additional hardware features that extend the concepts behind NoHype to increase performance and achieve an even greater level of security.

The remainder of the paper is organized as follows. In Section 2 we discuss the security threats that arise from the use of virtualization and formulate our threat model. In Section 3 we discuss the role of the virtualization layer. In Section 4 we propose our NoHype architecture which removes the virtualization layer. We then discuss the security benefits of moving to a cloud infrastructure which uses NoHype in Section 5. Then in Section 6 we take a look at currently available hardware technology and assess the possibility of realizing the NoHype architecture. Finally, we wrap up with related work in Section 7 and conclude in Section 8.

2. SECURITY THREATS

The problem we are addressing with the NoHype architecture is that of multi-tenancy in a hosted (public) cloud

[1] We will use the term *hypervisor* to mean hypervisor, VMM (virtual machine monitor) or other similar virtualization technology.

[2] A *compute unit* is a definition of a unit of computational resources that the customer purchases, e.g.: 1GHz CPU core, 2GB of RAM and 160GB of disk space.

environment. Essentially, the goal is to make running a virtual machine in the cloud as secure as running it in the customer's private facilities. In this section we briefly explain why this is not the case with today's virtualization-based cloud architecture. We also detail our threat model.

2.1 Concerns with Current Virtualization

Shown in Figure 1(a) is a diagram depicting the cloud customer's interaction with the cloud provider to control VM setup as well as an end user then using the service offered by the cloud customer. The cloud customer makes a request with a description of the VM (e.g., provides a disk image with the desired OS) to the cloud provider's *cloud manager*. The cloud manager chooses a server to host the VM, fetches the image from storage, and forwards the request to the control software running on the server to be handled locally. Once the cloud customer's VM is running, end users can begin using the cloud customer's service (end users can be external users, as in the case of web service, or internal customers, as in the case where the cloud is used to expand an internal IT infrastructure). Shown in Figure 1(b) is a generic diagram of one of the servers in today's infrastructures – consisting of a hypervisor along with several virtual machines. The virtual machine labeled "root context" is a special virtual machine which contains control software that interacts with the cloud manager and has elevated privileges that, for example, allow it to access devices and control the startup and shutdown of virtual machines. As such, it is considered part of the virtualization layer (along with the hypervisor)[3]. The virtual machines labeled VM1 and VM2 are the guest virtual machines running on the system and can be unmodified as the hypervisor provides the abstraction that they are running directly on the processor – in a hosted infrastructure, these would be the customer's virtual machines.

VM1 and VM2 should be completely isolated from one another. They should not be able to (i) inspect each other's data or software, (ii) modify each other's data or software, or (iii) affect the availability of each other (either by hogging resources or triggering extra work to be done by each other). As the virtualization layer is privileged software in charge of administering the physical system, a compromised virtualization layer can affect the running VMs. As it manages the virtual to physical mapping of memory addresses, confidential data may be exposed or the executable or data can be modified. As it manages the scheduling of VMs, a virtual machine can be interrupted (switching to the hypervisor's code) – exposing the current registers to inspection and modification, and control flow modification (e.g., by making the virtual machine return to a different location in the executable).

Unfortunately, securing the virtualization layer (hypervisor plus root context) is getting more difficult as hypervisors become more complex. Xen's hypervisor is about 100k lines of code and the dom0 kernel can be 1500k lines of code [12]. While the size of VMWare's solution is not publicly available, it is likely to match or exceed Xen's size. This complexity makes the current virtualization solutions difficult to verify and vulnerable to attacks – numerous vulnerabilities have already been shown in [1, 2, 3, 4, 5, 6, 7]. As the virtualization layer is complex, having many responsibilities as discussed in Section 3, we fully expect many new vulnerabilities to emerge.

To exploit one of these vulnerabilities, an attacker needs only to gain access to a guest OS and run software that can attack the hypervisor or root context (via the hypervisor) – since the guest OS interacts with both for many functions, there is a large attack surface. Getting access to a guest OS is simple as the malicious party can lease a VM directly from the cloud provider. Further, if the attacker is targeting a specific party (e.g., company A wanting to disrupt its competitor, company B), it can check whether its VM is located on the same physical server as the targeted victim's VM using network-based co-residence checks such as matching small packet round-trip times (between the attacker's VM and the victim's VM) and numerically close IP addresses [7].

To keep a guest VM from being able to exploit the vulnerabilities of the hypervisor, in the NoHype architecture we propose removing the hypervisor completely, as shown in Figure 1(c), by removing extraneous functionality not needed in the cloud computing scenario and by transferring some functionality to virtualization-aware hardware and firmware. We acknowledge that the hardware and firmware may not be completely free of bugs, however, we feel that the extensive testing and relatively non-malleable nature of the hardware and firmware configurations (when compared to software) makes it more difficult to attack. Of course, we will need to retain system management software used to start and stop VMs at the behest of the cloud manager. This software will be privileged, however to attack it will require going through the cloud manager first – guest VMs cannot directly invoke system management software in our NoHype architecture.

2.2 Threat Model

First, we assume the cloud provider is not malicious. The cloud provider's business model centers around providing the hosted infrastructure and any purposeful deviation from the stated service agreements would effectively kill its reputation and shut down its business. To that end, we assume that sufficient physical security controls are employed to prevent hardware attacks (e.g., probing on the memory buses of physical servers) through surveillance cameras and restricted access to the physical data center facilities.

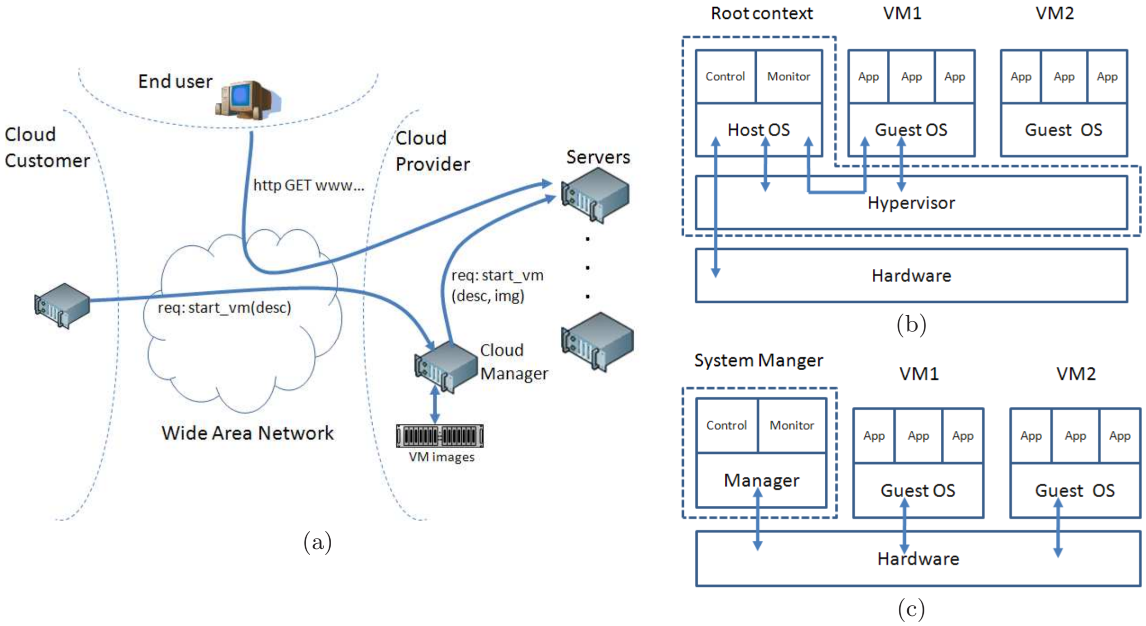
Second, we make no assumptions about security of guest operating systems. The cloud provider can restrict what OS the customer can run, but even that OS can have security holes. Therefore, we simply assume customers can run whatever software they desire.

Third, the security and correctness of the cloud management software is out of scope for this paper – we assume it is secure. The cloud management software runs on dedicated servers and is the interface that the cloud customers use to request and relinquish resources. From this, we also assume the system management software (system manager shown in Figure 1(c) and core managers) running on each server is also secure as the only communication it has is with the cloud management software. The guest VMs are isolated from it (no shared memory, disk or network), and do not interact with it.

3. THE VIRTUALIZATION LAYER'S ROLE

We propose the NoHype architecture which removes the virtualization layer in the multi-tenant server setting. To

[3] Note that the functionality of the "root context" could be included in the hypervisor for performance benefits. That has no impact on our argument.



(a)

(b)

(c)

Figure 1: (a) High-level view of the cloud customer's interaction with the cloud provider's management to start a VM and the end user's interaction with the service offered by the cloud customer (a web service in this case). (b) Generic virtualization of one server – arrows indicate interaction between the guest OS and hypervisor, host OS and the hypervisor, guest OS and the host OS (via the hypervisor), and the host OS and the I/O devices. (c) A server in the NoHype architecture after the removal of the hypervisor: the direct interaction between VMs and management software is removed.

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).

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, the hypervisor's scheduler routine is run to decide which VM to run next.

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*. In order to maximize utilization across all of the VMs, the hypervisor can coax one VM to page some memory to disk in order to be able to allocate that physical memory to another VM [11]. 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.

Emulating I/O Devices and Arbitrating Access to them : Access to the physical devices is essential as the I/O of the system is how a program interacts with the real world. Whether sending packets to a remote computer, or writing to storage, programs require I/O. As the machine 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.

Network Packet Processing (switching, NAT, and access control) : VM-to-VM communication is essential to enable communication between the services running inside the VMs. Since VMs can run on the same server, VM-to-VM communication may not need to go through the cloud provider's network. Because of this, modern virtualization technology typically runs a software Ethernet switch in the root context virtual machine with similar functionality as separate hardware switches would have [13].

Starting/Stopping/Migrating 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. The management software provides an interface to a central cloud manager which controls the creation and placement of virtual machines based on requests from customers, typically through a web interface. The management software may also be involved in live migration – the process of moving a running VM from one physical server to another while maintaining the appearance that the VM is continuously active to those that are interacting with it.

4. NOHYPE ARCHITECTURE: REMOVING THE HYPERVISOR

Our NoHype architecture removes the virtualization layer yet retains the management capabilities needed by cloud infrastructures. To do this, recall the major functions of the virtualization layer: arbitrating access to memory, CPU, and devices, providing important network functionality, and controlling the execution of virtual machines. Our architecture, shown in Figure 2, addresses each of these issues in order to remove the virtualization layer. Note that the cloud architecture remains unchanged, where the servers in Figure 1(a) are now NoHype-enabled servers.

The main point is that each of the guest VMs runs directly on the hardware without an underlying active hypervisor. Of course we cannot remove management software completely – there is the need for a management entity that can start and stop a virtual machine based on requests from the cloud provider's management software. Unlike today's virtualization architectures, in the NoHype architecture, once a VM is started it runs uninterrupted and has direct access to devices. The guest OS does not interact with any management software which is on the server and there are no tasks that a virtualization layer must do while the VM is running. In the following sub-sections we will discuss each of the roles of a typical hypervisor, and how the NoHype architecture removes the need for it.

4.1 CPU: One VM per core

In the NoHype architecture, each core can run only one VM. That is, cores are not shared among different guest VMs, which removes the need for the active VM scheduling done by the hypervisor. As an added benefit, dedicating a core to a single VM eliminates the potential software cache-based side channel which exists when sharing an L1 cache [14, 15].

Dedicating a core to each VM may not seem reasonable, as many associate virtualization with maxing out resources on each physical server to run as many VMs on it as possible. However, we believe that is counter to (i) the trend in computing towards multi-core, and (ii) the cloud computing model. The trend in processors is to increase the number of cores on the chip with each generation (as opposed to using clock frequency as the main source of performance improvements). Already, 8-core devices are available today [16] and there are predictions of 16-core devices becoming available in the next few years [17][4]. With each generation, the number of VMs a server can support would grow with the number of cores and sharing of cores to support more VMs will not be necessary.

Furthermore, we view running extra VMs to fill in for idle VMs (i.e., over-subscribing) to be counter to the model of cloud computing. The cloud infrastructure is dynamic in nature and the number of virtual machines needed by the customer can be scaled with the demand of the application. Therefore, idleness is handled by the customer by shutting down some of its virtual machines, instead of the cloud provider over-subscribing. While such over-subscribing can be done at a discount for customers who do not value security, it may not be necessary for customers who demand Secure Cloud Computing, especially with tens or hundreds of cores in future many-core chips.

4.2 Memory: Hardware support for partitioning and enforcing fairness

Unlike the processor, which is already moving toward divisible units, memory is still a unified resource. In the NoHype architecture, we propose using partitioning of physical memory. Here, we capitalize on the fact that modern servers are supporting more and more RAM – 256GB is not unheard of [18].

The ideal way to partition memory is to give each guest OS a view of memory where the OS has a dedicated and guaranteed fraction of physical memory (the *guest physical memory*) on the host system. Each VM can be assigned a different amount as decided by the customer when requesting a new VM to be started – any 'underutilization' is within each VM based on the customer requesting, and paying for, more than was needed. Given a portion of the memory, the OS will then be able to manage its own memory as it does today (managing the mapping from virtual memory locations to physical memory locations and swapping pages to/from disk as needed). Hardware support in the processor then performs the mapping between the *guest physical memory* address and the *host physical memory* address and restricts memory operations to the assigned range. In Figure 2, this responsibility falls to the multi-core memory controller (MMC) and the hardware page table mechanisms (inside each core) which will have built-in support for performing these re-mappings.

Not only must access to memory be partitioned, it must be fair. In the context of multi-tenant servers, rather than optimizing for memory bus utilization, the MMC must be designed for providing fairness among each of the VMs. As there can only be one VM assigned to a core, the MMC can instead provide fairness among each of the cores. This greatly simplifies the MMC as it can be designed for the number of cores the device has, rather than the number of VMs that may run on the system (which in traditional virtualization is undetermined and variable, and can be many more times than the number of cores). Including fairness in the multi-core memory controller and running one VM per core creates a system where one cloud customer's VM is greatly limited in how it can affect the performance of other cloud customers' VMs.

4.3 Devices: Per-VM virtualized devices and rate-limited I/O

An additional aspect of the physical system that needs to be partitioned is the access to I/O devices. In today's virtualization, operating systems interact with virtualized devices in the virtualization layer. This hides the details of the actual device and also allows the hypervisor to arbitrate access to the real device. Therefore, when the guest operating systems tries to access a device, this causes a switch over to the hypervisor and then to the root context VM. In the NoHype architecture, each guest operating system is

[4] We're referring to server class processors. Network processors and other specialized chips have on the order of 100 cores already.

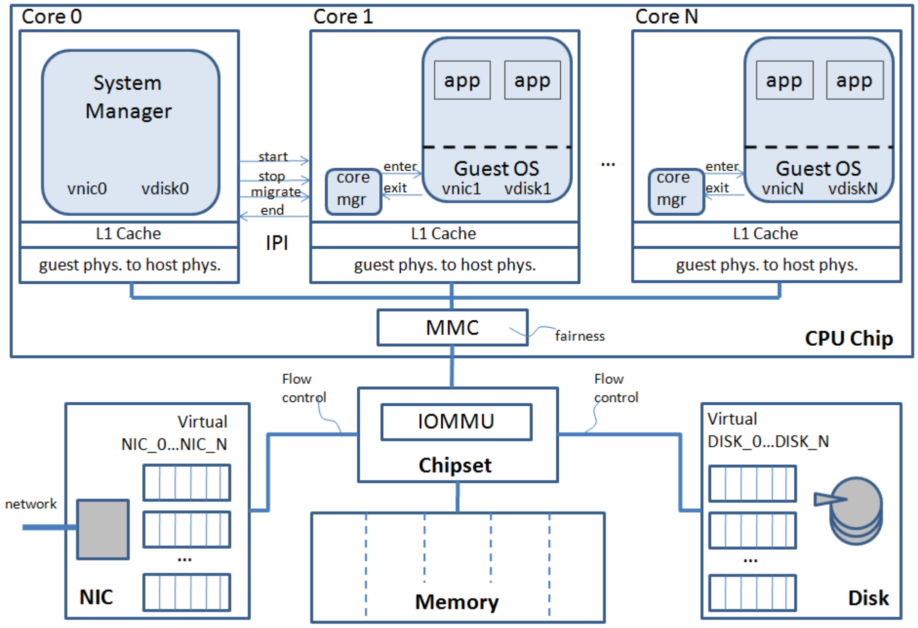


Figure 2: The NoHype system architecture.

assigned its own physical device and given direct access to it.

Of course, this relies on the assumption that there are enough devices to assign at least one per virtual machine. We believe that the view of multiple physical devices should be realized by the device itself supporting virtualization – that is, the device would be a single physical device, but tell the system that it is N separate devices. Each VM will interact only with the virtual device(s) assigned to it. As seen in Figure 2, a virtual device can have one or more queues dedicated to it. This forms the interface that is seen by the associated VM. The primary devices needed in a cloud computing scenario are the network interface card (NIC) and the disk. Other devices, such as graphics processing units (GPUs) could also be virtualized, thus removing a need for having N separate devices.

Providing this view requires some support in the MMC and I/O MMU[5]. For writes/reads to/from the device initiated by the cores, each device will be mapped to a different range in memory (as is done today) and each would be allowed to access only its memory ranges. This would enable the guest OS to interact directly (and only) with its assigned devices. From the device side, the I/O MMU would enforce the DMA to/from memory, so a device assigned to one core would only be able to access that core's memory range. For interrupts, since each VM will be assigned dedicated access to a given device, the interrupts associated with that device will be assigned to the guest virtual machine such that when an interrupt occurs, the guest OS handles it, not the virtualization layer.

One complication introduced by enabling direct access to devices is the bandwidth of the shared I/O bus (e.g., PCIe), is limited. When all device accesses go through the virtualization layer, the single piece of software can arbitrate access and prevent any single VM from overloading the shared bus. However, giving each VM the ability to access the devices directly means that there is no ability for software to prevent a VM from overloading the I/O bus. As such, in the NoHype architecture rate-limited access to each I/O bus is achieved via a flow-control mechanism where the I/O device controls the rate of transmission. Each VM's assigned device is configured with the rate at which the device can be accessed by the VM. The device uses this to limit the amount of data it sends over the peripheral bus and uses a feedback signal to limit the amount of data sent to it by the I/O MMU.

4.4 Networking: Do networking in the network, not on the server

In the NoHype architecture, we argue that the virtualization layer should be removed and instead its functionality should be provided through modifications to system components. For networking, this means that the Ethernet switches in the data center network should perform the switching and security functions, not a software switch in the virtualization layer. This is consistent with giving VMs direct access to the network interfaces – because they bypass the virtualization layer, the software Ethernet switch is also bypassed. Doing so has numerous benefits: (i) it simplifies management as it removes an extra type of switch and layer in the switch hierarchy, (ii) it frees up the processor on the server as it no longer has to perform, in software, Ethernet switching for an increasingly large number of VMs, and (iii) it allows the use of all of the features of the Ethernet switch, not just the ones also supported by the software switch.

Further, we argue that the software Ethernet switch is not doing anything special. Instead, it is merely used as an initial solution that enables using the Ethernet switches that are currently available. These switches are not designed

[5] I/O MMU - Input/Output Memory Management Unit is responsible for enforcing memory protection for transactions coming from I/O devices.

for virtualized infrastructures, but instead designed for enterprise networks. For example, because of its history as a shared medium, Ethernet switches drop packets that would be forwarded out of the same port as they arrived – a situation which would occur in virtualized infrastructures when two communicating VMs are located on the same server. Integrating support into hardware Ethernet switches for capabilities such as allowing a packet to be forwarded out of the same port as it was received would eliminate the need for the software switch.

While one may argue that requiring packets to go to the dedicated Ethernet switch has performance limitations, we note that this is only the case for the special situation of communication between two VMs located on the same server. In the cloud environment, the customers intended use of VMs is not known to the cloud provider. Making assumptions about the amount of traffic between co-resident VMs is equivalent to attempting to over subscribe the system, which we argue is not a good idea as the customer is paying for guaranteed resource usage. By not over subscribing, the peripheral bus bandwidth and bandwidth from the NIC to the top-of-rack switches must be provisioned to be able to support the guaranteed traffic bandwidth. With the software Ethernet switch, latency will be reduced for co-resident VM communication. However, it incurs the extra latency of going through an extra, lower performance (since it's in software), switch for *all* other packets. Instead, by bypassing the software Ethernet switch we are providing better average latency. While the cloud provider could attempt to maximize this situation by placing the VMs from a given customer on a single server, doing so comes at the cost of increasing the impact that a single server failure will have on that customer.

4.5 Starting/Stopping/Migrating Virtual Machines: Decouple VM management from VM operation

In the NoHype architecture we removed the hypervisor, yet we still need the ability to start and stop virtual machines on demand. To do this, we decouple the VM management from the VM operation – the management code is active before a VM is started and after it is stopped, but during the life of the VM, the guest OS never interacts with the management code on the core, or on the server (i.e., with the system manager running on a separate core). When a server starts up, one core is randomly selected as the bootstrap processor, as is done in today's multi-core systems. The code that starts executing on that core is the trusted *NoHype system manager*[6]. The system manager initially starts up in hyper-privileged mode to setup the server. It is then responsible for accepting commands from the cloud manager software (via its network interface) and issuing commands to individual cores to start/stop guest VMs via the inter-processor interrupts (IPIs). The sending, and masking, of IPIs is controlled through memory-mapped registers of the core's local APIC[7]. The memory management can be used to restrict the access to these memory regions to only software running in hyper-privileged mode, and therefore preventing VMs from issuing or masking IPIs. Upon receiving an IPI, the core will jump to a predefined location to begin executing the core manager code to handle the VM management. Figure 3 summarizes the actual procedures for starting and stopping VMs, with detailed description in the following paragraphs.

Starting a VM: Before a VM is started, the system manager must receive a command from the cloud manager. The instructions to the cloud manager are issued by the customer who specifies how many VMs and of what type he or she wants. The cloud manger then provides both a description of the VM (e.g., amount of memory it is assigned) and the location of the disk image to the system manager. Next, the system manager maps the to-be-assigned VM's memory and disk into its space to allow the manager to access the resources and initialize them. The disk image is then downloaded by the manager and stored on the local disk and the memory assigned to the core allocated to this VM is zeroed out. This brings in the guest OS image into the VM. Next, the to-be-assigned VM's disk and memory are un-mapped from the system manager's space so it no longer has access to them. Finally a 'start' inter-processor interrupt (IPI) is issued to the core where the VM is to start. Upon receiving a start IPI the core comes online and starts executing code which is stored at a predefined location. The code that executes is the *core manager* which starts running in the hyper-privileged mode and initializes the core (sets up the memory mapping and maps the virtual NIC and disk devices). To start the guest OS, the core manager performs a *VM exit* which switches the core out of the hyper-privileged mode and starts the execution of the guest OS from the image now stored locally on the disk. On bootup the guest OS reads the correct system parameters (e.g. the amount of memory that it has been assigned) and starts execution.

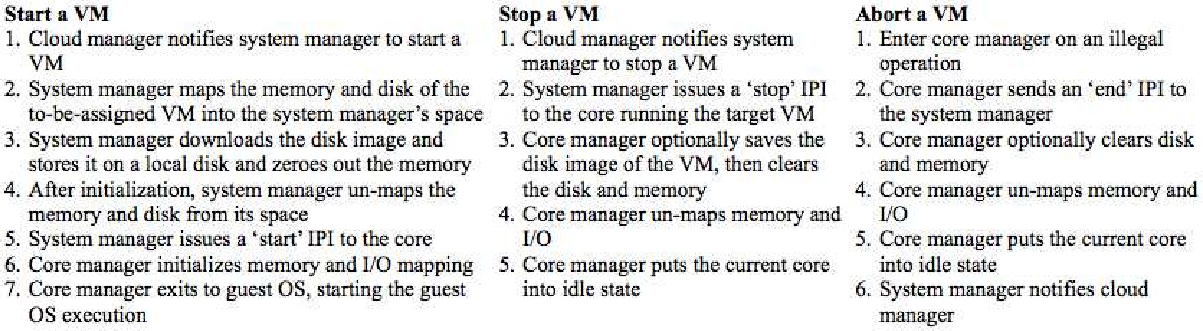
Stopping a VM: A guest OS can exit when a stop command is issued by the system manager (e.g., the system manager receives a message from the cloud manager that the customer does not need the VM anymore or the instance-hour(s) purchased by the customer have been used up). In this situation, the system manager sends a 'stop' IPI to the core running the VM that is to be shut down. This interrupt causes the core to switch to the hyper-privileged mode and jump to a predefined code location in the core manager's code. Next, the core manager optionally zeros out the memory space assigned to the VM, and potentially the assigned disk space if the customer's VM uploads its data to some storage before termination. The core manager also optionally saves the disk image of the VM, depending on the Service Level Agreement (SLA) for continuing service of this VM at a later time. Finally, the core manager puts the core in sleep mode (to wait for the next start IPI) and the system manager notifies the cloud manager of completion.

Aborting a VM: A guest OS can be aborted when the guest OS performs an illegal operation (e.g., trying to access memory not assigned to it)[8]. An illegal operation will cause a trap, which in turn causes entrance into the hyper-privileged mode and execution of code located at a predefined location in the core manager. At this point, the core

[6] The whole system manager does not have to be trusted, but for clarity of presentation and space reasons we will not explore issues of trust of the system manager here and hence we make this simplifying statement.

[7] APIC is the Advanced Programmable Interrupt Controller.

[8] Illegal operations also include the use of some processor instructions, which may require the OS to be altered. For example, we do not support nested virtualization, so any attempts to do so by the OS would be illegal.



Start a VM

1. Cloud manager notifies system manager to start a VM
2. System manager maps the memory and disk of the to-be-assigned VM into the system manager's space
3. System manager downloads the disk image and stores it on a local disk and zeroes out the memory
4. After initialization, system manager un-maps the memory and disk from its space
5. System manager issues a 'start' IPI to the core
6. Core manager initializes memory and I/O mapping
7. Core manager exits to guest OS, starting the guest OS execution

Stop a VM

1. Cloud manager notifies system manager to stop a VM
2. System manager issues a 'stop' IPI to the core running the target VM
3. Core manager optionally saves the disk image of the VM, then clears the disk and memory
4. Core manager un-maps memory and I/O
5. Core manager puts the current core into idle state

Abort a VM

1. Enter core manager on an illegal operation
2. Core manager sends an 'end' IPI to the system manager
3. Core manager optionally clears disk and memory
4. Core manager un-maps memory and I/O
5. Core manager puts the current core into idle state
6. System manager notifies cloud manager

Figure 3: Outline of steps used to start, stop and abort a VM.

manager sends an 'end' IPI to the system manager to inform the system manager that the VM exited abnormally. Optionally, the core manager can zero out the memory and the disk to prevent data leaks. The memory and I/O is un-mapped. The core is then put into sleep mode (waiting for a start IPI) and the system manager notifies the cloud manager of the aborted VM's status change.

Live Migration of a VM: A live migration operation is initiated by the cloud manager, which instructs the system manager on the source server to migrate a particular VM to a given target server. In a simplistic implementation, the system manager would send a 'migrate' IPI to a core on which the VM is running. The interrupt handler located in the core manager would stop the execution of the VM, capture the entire state of the VM, and hash and encrypt it. The system manager would then take the state, send it to the target server where the system manager would send an IPI to the core manager, which would check the hash and decrypt the state, re-start the new VM and continue execution.

Of course, this process can take a while and in order to minimize downtime, optimizations using iterative approaches have been developed [19]. In these approaches, a current snapshot is taken, but the VM is not stopped. When the state is done transferring, the difference from the current snapshot and the previous snapshot is sent. The process may be repeated until the difference is sufficiently small to minimize actual downtime. In NoHype, we have the memory management unit track which pages have been modified. This enables the system manager to periodically send an IPI to obtain only the differences (for the first time accessing it, the difference will be a comparison to when the VM started, so will be an entire snapshot). With this, we do not introduce a hypervisor which is actively involved. For each iteration, the system manager forwards any data to the target server – on the target server the system manager forwards data to the core manager, the core manager updates the memory, disk and other state it receives. The last step is for the core manager on the source server to send one last set of differences, shutting down the VM on the last 'migrate' IPI. After all the state is replicated on the target server, the system manager on the target sends a 'start' IPI to start the VM. It should be noted that while the downtime can be reduced from what would be seen with the simplistic approach, it cannot be eliminated altogether. In Xen, this may range from 60ms for an ideal application to 3.5s for a non-ideal case [19]. The nature of cloud computing makes it such that (i) how long the downtime will actually be for a given VM cannot be known, and (ii) the downtime the customer is willing to tolerate is not known. As such, an alternate approach would be to involve the customer in the process, enabling them to gracefully 'drain' any ongoing work or perform migration themselves [20].

For all four actions, during the runtime of the guest OS, the guest OS does not invoke either the core manager or the system manager. Even during a live migration, no action the guest OS performs causes any management software to run – each iteration is initiated by the system manager. Hence, the guest OS has no opportunity to directly corrupt these trusted software components. Interaction with the cloud manager is from servers external to the cloud infrastructure (i.e., the customer's server). Securing this interaction is not the focus of this paper.

5. SECURITY BENEFITS OF NOHYPE

Because of the numerous benefits that hosted cloud infrastructures provide, many organizations want to use the cloud. However, concerns over security are holding some of them back. The NoHype architecture targets these concerns, creating an architecture where customers are given comparable security to running their VMs in their own virtualized infrastructure – and even *improved* security when considering the extra physical security and protection against malware that cloud providers can provide.

To achieve a comparable level of security, no VM should be able to (i) affect the availability of another VM, (ii) access the data/software of another VM (either to read or modify), or (iii) learn confidential information through side channels. Note that this does not mean that the cloud customer is completely protected, as vulnerabilities in the cloud customer's own applications and operating system could still be present – end users can attack a vulnerable server independent of whether it is running in the cloud or not.

Availability : Availability can be attacked in one of three ways in current hypervisor-based virtualization architectures– (i) altering the hypervisor's scheduling of VMs, (ii) interrupting a core running a VM, or (iii) performing extraordinary amounts of memory or I/O reads/writes to gain a disproportionate share of the bus and therefore affect the performance of another VM. By dedicating a core to a single VM and removing the hypervisor from making any scheduling decisions, we eliminate the first attack. With hardware masking for inter-processor and device interrupts, there is no possible way to interrupt another VM, eliminating the second attack. Through additions to the mutli-core memory controller for providing fairness [21] and through the chipset to rate-limit access to I/O, we eliminate the third attack. As such, with the NoHype architecture, a VM has no ability to disrupt the execution of another VM.

Confidentiality/integrity of data and software : In order to modify or inspect another VM's software or data, one would need to have access to either registers or physical memory. Since cores are not shared and since there is no hypervisor that runs during the entire lifetime of the virtual machine, there is no possible way to access the registers. Memory access violations are also mitigated. Since the NoHype architecture enforces memory accesses in hardware, the only way a VM could access physical memory outside of the assigned range would be to alter the tables specifying the mapping of *guest physical* addresses to *host physical* addresses. To do so would require compromising the system manager software and altering the code performing the start/stop functions. This would first require compromising the cloud manager, which we assume is trusted, as the system manager only interacts with the cloud manager and the core managers, and is isolated from the guest VMs.

Side-channels : Side-channels exist whenever resources are shared among multiple pieces of software. Ignoring any side-channels that are based on physical analysis (e.g., examining power usage), which falls outside of our threat model, side-channels are typically based on the timing of operations (e.g., hits in caches are shorter than misses, so one can determine if a particular cache line was accessed by timing an access to a specific memory location). While completely eliminating side-channels is a near impossible feat for a shared infrastructure, reducing them to be of little use is possible. In the NoHype architecture, since L1 caches are not shared, some of the most damaging attacks to date have been eliminated (using cache-based side-channel attacks to infer a cryptographic key) [22]. Since NoHype provides fair access to memory [21] and rate-limited accesses to I/O, a bandwidth-based side channel is likely very limited in each of those cases as well.

6. CURRENT HARDWARE SUPPORT

In Section 4, we presented our NoHype architecture. In this section, we examine current technology and whether it is possible to actually realize NoHype. A lot of hardware extensions, both in the processors and in the devices, have been introduced in recent years which aid computing in the virtualized model. Interestingly, while these extensions are mostly designed to address the performance overhead, we are able to leverage these in the NoHype architecture that we propose. We will present extensions found in Intel products – similar solutions are available from AMD and NoHype should be implementable on AMD platforms as well.

6.1 Isolating Each Core

As multi-tenancy requires running multiple virtual machines on the same processor, they need to share the processor resources. In order to remove the need for the hypervisor to schedule each VM, the NoHype architecture assigns a single VM per core. To fully isolate each VM from other VMs, however, we need to be able to restrict a core running one VM from sending IPIs to other cores. This can be achieved through use of the advanced programmable interrupt controller (APIC). Modern x86 processors have an interrupt mechanism based on the APIC. As shown in Figure 4, each core has a local APIC and there is a single I/O APIC for the entire system.

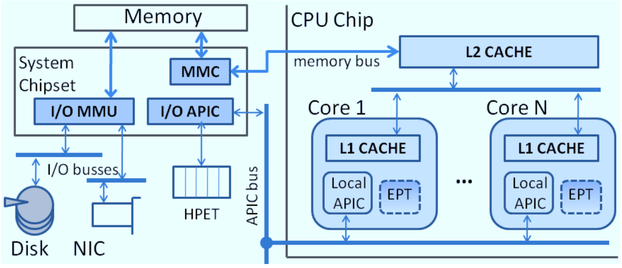


Figure 4: Selected parts of a modern x86 architecture which are relevant to NoHype.

The APIC can be used by a core to send an interrupt to another core. These IPIs[9] are generated by programming the interrupt command register (ICR) in the core's local APIC. By writing to this register, a message will be sent over the APIC bus with the destination field of the ICR indicating which core should receive the interrupt. While one VM can attempt to interrupt another VM with this mechanism, each core can mask out maskable interrupts that may be sent from other cores, to prevent them from interrupting the core.

6.2 Memory Partitioning

A key to isolating each virtual machine is making sure that each VM can access its own *guest physical memory* and not be allowed to access memory of other VMs. In order to remove the virtualization layer, we must have the hardware enforce the access isolation. Here, we capitalize on the paging protection available in modern processors, along with advances in providing these protections to each VM.

The Extended Page Tables (EPT) mechanism supported by VT-x technology from Intel can be used to enforce the memory isolation. The EPT logically adds another layer to the familiar virtual-to-physical memory translation page tables. The OS running on a core manages the translation from guest virtual to guest physical addresses, using the familiar mechanisms (e.g., CR3 register is used to set the page directory location). The EPT mechanisms are then used to translate from the guest physical to host physical address – the set of page tables which define this mapping are initialized by the core manager. Once initialized, the translation does not have to be modified throughout the lifetime of the OS instance running on a given core. A hardware page table walker uses the tables to automatically perform the translation. Access to a virtual address not currently in the guest physical memory will cause a page fault that is handled by the guest OS – a hypervisor layer is not needed. Access to memory outside of the guest physical address range preassigned to the VM will cause a new fault which is handled by the core manager, which causes the VM to be killed – as any action that would exit to a virtualization layer is assumed by NoHype architecture to be due to a misbehaving guest OS.

An open issue is the memory fairness and preventing cores

[9] The "processor" in "inter-processor interrupt" actually refers to a processor core.

from interfering with each other through use of the the memory subsystem. Currently, a processor core can hog the memory bandwidth, based on current memory controllers' scheduling algorithms which optimize overall memory performance, rather than being fair to all cores. We propose using a solution similar to one presented by Mutlu, et al., [21] whereby the memory controller is modified to help achieve a more fair sharing of memory bandwidth.

Another open issue is the optimal support for live migration that is completely transparent to the customer (involving the customer requires no support from the system). The simplistic *stop-and-copy* approach is fully supported. However, in order to use optimizations which iteratively copy differences during execution requires tracking dirty pages. Current processors do not support this, so software would be needed. In this case, each time a page is accessed, an exception would occur and be handled by the core manager which would simply mark a bit in a table. As this interaction is extremely limited (no information is passed and the required function is extremely simple), the vulnerability that is opened is very limited (though, still not ideal).

6.3 I/O Devices

The NoHype architecture relies on each virtual machine having direct access to its own physical device(s). Fortunately, I/O performance is an important factor in overall system performance, and as such, technologies are available to make this a reality.

6.3.1 Direct Access to I/O Devices

In the NoHype architecture the virtual machine must be able to access its own devices without going through the virtualization layer. This involves three aspects: (i) the VM being able to DMA to the device, (ii) the device being able to DMA to the VM, and (iii) the device being able to send an interrupt to the VM.

For the VM being able to DMA to the device, the memory partitioning discussed in Section 6.2 is the enabling technology. Each VM is assigned a device, with each device being assigned a memory mapped I/O range. The guest physical to host physical memory mapping enables VMs to directly interact with memory mapped I/O. The fact that the guest VM can only write to particular physical memory ranges, enforced by the hardware, means that each VM can write only to its device.

In the reverse direction, the I/O device sending data to the VM, requires support from the chipset which sits between the device and the system memory. Here the I/O MMU provides protection by creating multiple DMA protection domains. These domains are the memory regions that a given device is allowed to DMA into. This would correspond to the assigned physical memory that the VM was given. The chipset enforces this by using address-translation tables. When a device attempts to write to a memory location, the I/O MMU performs a lookup in its address-translation tables for access permission of that device. If the device is attempting to write to a location outside of the permitted range, the access is blocked.

The final aspect of interacting with devices without the involvement of the virtualization layer is the ability for interrupts from a device to be directly handled by the virtual machine associated with that device. For this, the MSI (Message Signaled Interrupts) and extension MSI-X specification [23] serve just this purpose. With MSI-X, a device can request an interrupt by writing to a particular memory address. The value written to that address indicates which interrupt is to be raised. By setting up the APIC tables, a given interrupt can be directed to a specific core on the processor (via its local APIC). The memory protection mechanisms can be used to block devices from writing certain memory locations, consequently limiting which device can interrupt which core.

6.3.2 I/O Device Virtualization

While it is possible to realize each VM having its own device by having multiple physical devices, as the number of cores (and therefore the number of VMs) grows, this will become impractical. The advantage we have here when considering the hosted cloud infrastructure setting is that there is only a small subset of devices that are actually of interest: the network interface card, the storage device, and a timer, but not, for example, a monitor or keyboard. Here, we can take advantage of a standard specified by PCI-SIG called single root I/O virtualization (SR-IOV). SR-IOV enables a single device to advertise that it is actually multiple devices, each with independent configuration and memory space.

To do this, the devices present two types of interfaces: one for configuration and the other for regular device access. The system software can use the configuration interface to specify how many virtual devices the physical device should act as and the options for each virtual device. Each virtual device is assigned a separate address range, enabling the system manager to map a virtual device to a particular virtual machine. The guest VM can then interact directly with the device through the regular device interface presented by the physical device. Only the system manager has the privileges to use the configuration interface and the VMs can only access the interfaces assigned to them.

Network: Intel has a network interface card that supports SR-IOV [24]. To distinguish between each of the virtual devices, the NIC has multiple queues on it. The NIC performs classification on the packets to direct each packet to the correct queue. Then, because of the memory partitioning, only the OS on the appropriate core can read directly from that virtual device, or write to it..

Storage: Unfortunately, there are currently no available storage devices which support the notion of virtual devices. LSI, however, recently announced a demonstration of an SR-IOV capable storage controller [25]. Additionally, more should be coming as disk drives already have a lot of functionality in the form of firmware on the drive itself – for example, reordering requests to optimize read/write performance. Furthermore, there are disk devices such as from Fusion-io which have a PCIe interface (and therefore can implement SR-IOV) and a programmable FPGA on board (and therefore can be upgraded). Until the LSI chip is integrated into disk drives or firmware is modified, multiple physical disk drives can be used. Server motherboards have already multiple SATA (Serial Advanced Technology Attachment) connections. When used in combination with network storage, this provides an acceptable, though not ideal, solution until SR-IOV enabled disk drives become available.

Timers: Timers are an essential part of operating systems and many programs which perform periodic tasks (such as scheduling processes). Each local APIC contains a 32-bit timer that is software configurable, and therefore provides each VM with its own timer. However, higher-precision timers require an external chip – the HPET (High Precision Event Timer). The HPET communicates with the I/O APIC, which as previously mentioned can be configured to deliver (timer) interrupts to a specific core via its local APIC. The HPET specification allows for up to 8 blocks of 32 timers and the HPET configuration is memory mapped – the map for each block can be aligned on a 4 kilobyte boundary so paging mechanisms can easily be used to restrict which core has access to which block of timers. In this way, the core can have full control over its timers which are not affected by other VMs.

Both the networking and disk devices' bandwidth usage need to be shared among the VMs. If a VM is able to use a disproportionately large amount of the I/O bandwidth, other VMs may be denied service very easily. Fortunately, the PCI Express (PCIe) interconnection network uses credit-based flow control. Each device has an input queue and senders consume tokens when sending packets. When the queue in the device associated with the sending core becomes full, the sender no longer has tokens for sending data. A device, e.g. the NIC, with virtual queues will take packets from the queues in round-robin fashion and return tokens to the sender. If a VM fills up its assigned queue, it will not be able to send more packets which will allow other VMs to regain their share of the bandwidth.

6.4 Virtualization aware Ethernet Switches

In the NoHype architecture, the networking is performed by networking devices and not by the software running on the server. This requires the Ethernet switches that are connected to the servers (e.g., the top-of-rack switches) to be virtualization aware. The switches need the abstraction that they have a connection to each virtual interface, rather than the single connection to the NIC shared across multiple virtual machines. As the market for virtualized infrastructures (not just hosted cloud infrastructures) is potentially very large, commercial Ethernet switch vendors are moving in this direction already. The Cisco Virtual Network Link (VN-link) [26] technology is supported in Cisco Nexus switches [13] where it is used to simplify the software switch and offload switching functionality to a physical switch, but this requires some changes to the Ethernet format. In contrast, HP has proposed Virtual Ethernet Port Aggregator (VEPA) [27], which does not require changes to the Ethernet format, but instead overcomes the limitation of not being able to forward a packet out of the same port it was received. Commercial availability of hardware switches which support VEPA are forthcoming, and Linux has support for VEPA for software based switches [28]. (not suitable for use in the data center, but useful for experimentation).

7. RELATED WORK

Virtual machine monitors and hypervisors have been studied extensively. Some of the work relating specifically to securing the virtualization layer in a server or "cloud" setting includes work by Santos, et al., [29] and sHype [30] from IBM. Santos, et al., presented a solution which uses the Terra hypervisor [31]. The sHype is a secure hypervisor architecture whose goal is to provide a secure foundation for server platforms. Rather than attempt to secure the virtualization layer, we instead removed it.

We should note that trusting the virtualization layer usually comes with the disclaimer that the virtualization layer must be minimal (i.e., minimizing the Trusted Computing Base, TCB). A number of projects have worked on minimizing the TCB (for example Flicker [32]) which attests to the importance of having the TCB as small as possible. None, however, went so far as to completely remove the hypervisor that actively runs under the OS, as we have.

Finally, hardware architectures have been proposed to enable running applications without trusting the operating system [33, 34, 35, 36, 10]. While these could perhaps be extended, or applied, to support running a trusted virtual machine on an untrusted hypervisor, we instead took an approach which proposes minimal changes to the processor core architecture along with modifications to system peripherals and software.

8. CONCLUSIONS

While cloud computing has tremendous promise there are concerns over the security of running applications in a hosted cloud infrastructure. One significant source of the problem is the use of virtualization, where an increasingly complex hypervisor is involved in many aspects of running the guest virtual machines. Unfortunately, virtualization is also a key technology of the cloud infrastructure enabling multi-tenancy and the ability to use highly automated processes to provision infrastructure. Rather than focus on making virtualization more secure, we remove the need for an active virtualization layer in the first place.

We presented the NoHype architecture which provides benefits on par with those of today's virtualization solutions without the active virtualization layer. Instead, our architecture makes use of hardware virtualization extensions in order to remove any need for a virtualization layer to run during the lifetime of a virtual machine. The extensions are used to flexibly partition resources and isolate guest VMs from each other. Key aspects of the architecture include (i) running one VM per core, (ii) hardware enforced memory partitioning, and (iii) dedicated (virtual) devices that each guest VM controls directly without intervention of a hypervisor. We also discussed the fact that current processor and I/O devices support much of the functionality required by the NoHype architecture. While there are clearly more architectural, implementation, software, performance and security issues to be resolved, we believe that the architecture is not hype but is a very viable architecture that can be implemented in the near future. We also hope that this paper will stimulate further research into many-core architectural features, at both the processor and system levels, that support using the many cores for security as in our NoHype architecture, rather than just for performance and power improvements.

9. ACKNOWLEDGMENTS

This work was supported in part by NSF CCF-0917134 and by research gifts from Intel and IBM. We would like to thank the anonymous ISCA reviewers, and also James Hamilton and Steven Ko for their feedback.

10. REFERENCES

[1] K. Kortchinsky, "Hacking 3D (and Breaking out of VMWare)," *BlackHat USA*, 2009.

[2] Anonymous, “Xbox 360 hypervisor privilege escalation vulnerability,” 2007. http://www.h-online.com/security/news/item/Xbox-360-hack-was-the-real-deal-732391.html.

[3] “CVE-2007-4993: Xen guest root can escape to domain 0 through pygrub,” 2007. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4993.

[4] “CVE-2007-5497: Vulnerability in XenServer could result in privilege escalation and arbitrary code executionr,” 2007. http://support.citrix.com/article/CTX118766.

[5] R. Wojtczuk, “Subverting the Xen hypervisor,” *BlackHat USA*, 2008.

[6] “CVE-2008-2100: VMware Buffer Overflows in VIX API Let Local Users Execute Arbitrary Code in Host OS.” http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2100.

[7] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud! Exploring information leakage in third-party compute clouds,” in *Computer and Communications Security(CCS)*, 2009.

[8] F. Gens, “IT Cloud Services User Survey, pt.2: Top Benefits & Challenges,” Oct. 2008. http://blogs.idc.com/ie/?p=210.

[9] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: Research problems in data center networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2009.

[10] D. Champagne and R. B. Lee, “Scalable architectural support for trusted software,” in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, January 2010.

[11] C. A. Waldspurger, “Memory resource management in VMware ESX server,” in *5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[12] D. G. Murray, G. Milos, and S. Hand, “Improving Xen security through disaggregation,” in *VEE ’08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2008.

[13] “Cisco Nexus 1000V Series Switches,” 2009. http://www.cisco.com/en/US/products/ps9902/index.html.

[14] Z. Wang and R. B. Lee, “New cache designs for thwarting software cache-based side channel attacks,” in *34th International Symposium on Computer Architecture (ISCA)*, pp. 494 – 505, 2007.

[15] Z. Wang and R. B. Lee, “A novel cache architecture with enhanced performance and security,” in *Proceedings of the 41st. Annual IEEE/ACM International Symposium on Microarchitecture (Micro-41)*, pp. 88–93, December 2008.

[16] “Intel Previews Intel Xeon Nehalem-EX Processor,” 2009. http://www.intel.com/pressroom/archive/releases/20090526comp.htm.

[17] A. Shah, “AMD plans 16-core server chip: The Interlagos chip will be released in 2011,” *Infoworld*, April 2009.

[18] “Dell PowerEdge R905 server product details,” 2009. http://www.dell.com/us/en/enterprise/servers/pedge_r905/pd.aspx?refid=pedge_r905.

[19] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines,” in *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[20] M. A. Kozuch, M. Kaminsky, and M. P. Ryan, “Migration without virtualization,” in *12th Workshop on Hot Topics in Operating Systems (HotOS)*, 2009.

[21] O. Mutlu and T. Moscibroda, “Stall-time fair memory access scheduling for chip multiprocessors,” in *MICRO 2007*, 2007.

[22] D. J. Bernstein, “Cache-timing attacks on AES,” in *University of Illinois at Chicago Tech Report*, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[23] “PCI Local Bus Specification, 3.0 edition,” 2004.

[24] “Intel 82599 10 gigabit ethernet controller,” 2009. http://download.intel.com/design/network/prodbrf/321731.pdf.

[25] “LSI to Demonstrate Industry’s First Single-Root I/O Virtualization Solution at Intel Developer Forum,” 2009. http://www.lsi.com/news/product_news/2009/2009_09_22.html.

[26] “Cisco VN-Link: Virtualization-Aware Networking,” 2010. http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns892/ns894/white_paper_c11-525307_ps9902_Products_White_Paper.html.

[27] “Edge Virtual Bridging with VEB and VEPA,” May 2009. http://www.ieee802.org/1/files/public/docs2009/new-hudson-vepa_seminar-20090514d.pdf.

[28] “Net/bridge: add basic VEPA support,” June 2009. http://lwn.net/Articles/337547/.

[29] N. Santos, K. P. Gummadi, and R. Rodrigues, “Towards trusted cloud computing,” in *Workshop On Hot Topics in Cloud Computing (HotCloud)*, 2009.

[30] IBM, “sHype - Secure Hypervisor.” http://www.research.ibm.com/secure_systems_department/projects/hypervisor/.

[31] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, “Terra: A virtual machine-based platform for trusted computing,” *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 193–206, 2003.

[32] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, “Flicker: An Execution Infrastructure for TCB Minimization,” in *ACM European Conference in Computer Systems (EuroSys)*, 2008.

[33] G. E. Suh, C. W. O’Donnell, I. Sachdev, and S. Devadas, “Design and implementation of the AEGIS single-chip secure processor using physical random functions,” in *Proc. ISCA*, June 2005.

[34] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang, “Architecture for protecting critical secrets in microprocessors,” in *Proc. ISCA*, June 2005.

[35] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, “Architectural support for copy and tamper resistant software,” in *Proc. ASPLOS*, November 2000.

[36] J. Dwoskin and R. B. Lee, “Hardware-rooted trust for secure key management and transient trust,” in *ACM Conference on Computer and Communications Security (CCS)*, October 2007.