Architectural Support of Multiple Hypervisors over Single Platform for Enhancing Cloud Computing Security

Weidong Shi
University of Houston
Houston, TX 77004, USA
larryshi@ymail.com

JongHyuk Lee
University of Houston
Houston, TX 77004, USA
jonghyuk.lee@daum.net

Taeweon Suh
Korea University
Seoul, South Korea
suhtw@korea.ac.kr

Dong Hyuk Woo
Intel Labs
Santa Clara, CA 95054, USA
dong.hyuk.woo@intel.com

Xinwen Zhang
Huawei America R&D Center
Santa Clara, CA 95050, USA
xinwenzhang@gmail.com

ABSTRACT

This paper presents MultiHype, a novel architecture that supports multiple hypervisors (or virtual machine monitors) on a single physical platform by leveraging many-core based cloud-on-chip architecture. A MultiHype platform consists of a control plane and multiple hypervisors created on-demand, each can further create multiple guest virtual machines. Supported at architectural level, a single platform using MultiHype can behave as a distributed system with each hypervisor and its virtual machines running independently and concurrently. As a direct consequence, vulnerabilities of one hypervisor or its guest virtual machine can be confined within its own domain, which makes the platform more resilient to malicious attacks and failures in a cloud environment. Towards defending against resource exhaustion attacks, MultiHype further implements a new cache eviction policy and memory management scheme for preventing resource monopolization on shared cache, and defending against denial of resource exploits on physical memory resource launched from malicious virtual machines on shared platform. We use Bochs emulator and cycle based x86 simulation to evaluate the effectiveness and performance of MultiHype.

Categories and Subject Descriptors

D.4 [**Operating Systems**]

Keywords

Virtualization, Architecture, Security, Scalability

1. INTRODUCTION

Cloud computing is emerging as a viable alternative to premise-based deployment of hardware and software systems. The economy of scale and elasticity offered by cloud computing has garnered rapid adoption for increasingly dynamic and competitive business climate. As a consequence, cloud computing is quickly altering the landscape of the information technology service industry. Virtualization plays a critical role in cloud computing by multiplexing the resources and computing power of a single platform to multiple logical platforms. The development of virtualization technology has turned traditional software into *virtual appliances* and allows software and their execution environment to be rapidly deployed and delivered as services in ways that are both massively scalable and elastic. According to IDC's analysis, cloud services will be in the order of $44.2bn in 2013 [8].

Virtualization has existed long before the emergence of cloud computing. However, with the light of recent advance on low cost many-core processors, virtualization has made cloud computing economically viable. Specifically, many-core processors have increased virtualization density to the point where large numbers of virtual servers can be ran concurrently on a single physical server. In foreseeable future, the number of processor cores in a single processor will continue to double steadily [11]. Therefore, we will reach the era of *hyperscale* virtual server consolidation where hundreds or even thousands of virtual servers can be packed on a single many-core based physical server. This will enable virtualization based computing at epic scale.

On the other side, although cloud computing holds great potential and promises, security is one of the main challenges and deficiencies in today's cloud environment. Not surprisingly, the characteristics of multi-tenancy and shared resources introduce new risks and threats to any resources on cloud platform. Potential risks include failure of separation mechanisms for storage, memory, routings between different tenants, and hypervisor subversion [30]. Further threats come from the possibilities of attacks that "escape" from a guest virtual machine (VM) and being able to inject codes into the host system or other VMs [16, 22]. Public consent and study [10, 17] have shown major security concerns, including the reluctance to deploy virtual machines on shared physical servers (which run against the fundamental cloud computing principles of resource sharing and on-demand provisioning), potential leak and disclosure of confidential and proprietary information to third parties, and compromising of co-located virtual machines. These concerns are well justified by identified and potential vulnerabilities associated with commodity hypervisors and virtual machine systems on shared platforms [25, 7, 16, 22, 30]. Due to those concerns, many data center customers demand their services be hosted by dedicated servers physically isolated from other customers' servers. In the near future, we can expect to see many new security exploitations on cloud environment towards platforms and user information.

In line with these trends and challenges, we propose *MultiHype*, a poly-hypervisor architecture to improve the dependability, scalability, and security of many-core based cloud servers. Comparing with today's mono-hypervisor (MonoHype) based systems, MultiHype supports running multiple hypervisors or VMMs (virtual machine monitors) on a single physical platform, in turn, each of which can execute

CF'12, May 15–17, 2012, Cagliari, Italy.

multiple VMs. By leveraging many-core based cloud-on-chip processor architecture, this extra abstraction provides strong isolation between physical resources managed by individual hypervisor *realms*. As a direct consequence, the vulnerabilities of one hypervisor or a VM within a hypervisor can be confined within its own domain, which makes platform-wide attacks much harder. With careful design on physical separation of CPUs, memory, storage, and I/O devices, MultiHype can achieve more dependable, secure, and scalable cloud server platform than today's MonoHype platform, which fits the requirement of hyper-scale virtual server consolidation.

Outline: Next section analyzes and summarizes security risks, threats, and attacks on existing cloud servers. In Section 3 we present the architecture and design of MultiHype. We then illustrate the evaluation methodology of MultiHype and result analysis in Section 4 and 5, respectively. Section 6 summarizes related work of this paper and Section 7 concludes this paper.

2. HYPERVISOR RELATED THREATS IN CLOUD COMPUTING

In cloud computing environment, virtual machines from different cloud customers share a single physical server and hypervisor. Virtualization offers "layered defense" for system security, usually by assuming that a malicious attacker who controls or penetrates one guest virtual machine cannot compromise the underlying system and other virtual machines. This should not always be taken for granted. Previous studies have demonstrated the vulnerabilities and real attacks that determined attackers can exploit hypervisor vulnerability, and consequently compromise services of co-located cloud users [25, 7, 16, 22, 30, 24]. We summarize several risks and threats of virtualized platform in cloud computing environment as follows.

2.1 Hypervisor Vulnerabilities

On a cloud server platform, virtual machines from different customers sit above a common hypervisor that manages both the physical hardware resources and customer resources. Like any other software layer, a hypervisor can have vulnerabilities and is prone to attacks or unexpected failure. Commodity hypervisor has significantly grown in functions and features, and thus in code size. These make them look closer to a real operating system (OS) with large trusted computing base (TCB), and increasing design and implementation vulnerabilities. Therefore their isolation and security functions might be compromised by attacks from guest OS [13]. An attacker can compromise a hypervisor by hacking it from inside a guest virtual machine and exploit all the guests. Hypervisor layer attacks are very attractive. In a MonoHype system, the hypervisor fully controls the physical resources and all guest virtual machines that run on top of it.

Past few years have seen a number of successful hypervisor subversions [25, 7, 16, 22, 30, 24]. King et al. [15] described the concept of a virtual machine-based rootkit and demonstrated the subversion of VMWare and VirtualPC using hypervisor rootkit SubVirt. Blue Pill [24] is a rootkit that can trap a running native OS into a guest virtual machine "on-the-fly" with hardware-assisted virtulization technology such as Intel VT-x or AMD Pacifica. In [22], the author investigated several popular x86 based virtual machine implementations and tested whether the assumed hypervisor security and virtual machine isolation can be taken for granted. The authors performed hypervisor stress tests by injecting random instructions and I/O activities to the hypervisor from a guest virtual machine. The results identified vulnerabilities in all popular virtual machine implementations for x86 architecture in use today. If exploited, a vulnerable VMM can be subverted to execute arbitrary code on the host with the privileges of the VMM process. In addition, an exploit from a virtual machine guest could cause VMM to terminate unexpectedly or trigger an infinite loop that prevents the host from performing normal administration operations for other virtual machines.

When a hypervisor is subverted, an attacker can escape the isolation between different customers. For example, a documented attack on VMWare [25] allows "guest to host escape". After a hypervisor is subverted, an attacker may take control of other virtual machines running on the same hypervisor or gain access to the data contained inside them. Furthermore, an attacker may manipulate resource allocation; reduce resources assigned to other virtual machines and as a consequence cause denial of service.

2.2 Weak Separation between VMs

Cloud computing infrastructures mostly rely on architectural designs to separate physical resources in a logical manner such as to share computing capacity, storage, and network among multiple virtual machines and therefore multiple customers. On a many-core computing platform, hundred or even thousand of virtual machines may share the same physical platform. Failure of resource separation between different tenants could lead to potential devastating results such as unauthorized access to shared resources, provoking denial of services by manipulating resources allocated to other customer's virtual machines or terminating other customer's running virtual machines, and side-channel data leakage. In [23], the authors illustrated the steps for accessing confidential information from running EC2 instances and demonstrated side-channel exploits.

2.3 Resource Exhaustion

Cloud services acquire resources in on-demand manner. In multi-tenancy working environment, malicious attackers may trigger resource exhaustion and cause denial-of-resources attacks to other users' virtual machines. Shared resources with capacity limitation include memory, storage, I/O bandwidth, networking buffers, CPU, etc. If an attacker can trigger the allocation of these limited resources, but the number or size of the resources is not controlled, the attacker could cause a denial of service by consuming all available resources on a physical platform. For example, a memory exhaustion attack against an application could slow down the application as well as its host OS. A malicious customer may run mischievous guest virtual machines that use certain resources intensively. For example, a virtual machine can deliberately trigger a lots of interrupts or generate switches between virtual machine and hypervisor at extremely high frequency.

3. ARCHITECTURE AND DESIGN

3.1 Requirements

Comparing with MonoHype systems [3, 19], a MultiHype system in ideal scenario should satisfy the following requirements.

- Multiple and separated hypervisors that can scale with large scale many-core based platform (e.g., hundreds of cores platform) and support of hyper-scale virtual server consolidation for multiple customers;

- Two-tiered resource allocation and isolation mechanism: resources such as CPUs and memory are first allocated and partitioned among hypervisors and then for each hypervisor, the resources are shared among guest virtual machines; and

- Security breach compartmentalization: an architecture capable of preventing security breach from spreading to other customer's hypervisors and virtual machines.

One desirable feature to support these requirements is that different hypervisors share minimal physical resources, thus the normal function of a hypervisor requires little or no interaction from other hypervisors. With this, an attack on one hypervisor by a malicious customer or attacker

Table 1: Comparison of Three Frameworks

	Multiple Hypervisors Single Physical Platform	Single Hypervisor Single Physical Platform	Distributed Hypevisors Multiple Physical Platforms
Hypervisor Vulnerabilities	Confined	Not confined	Not confined
Risks of Resource Monopolization	Low	High	High
Hypervisor Single Point-of-Failure	No or very limited	Yes	Yes
Shared Hypervisor States	Guests of the same realm/customer	All guests	Across physical servers
Hypervisor Creation	On-demand	Boot time	Boot time
Examples	MultiHype	All MonoHype based including [5]	3leaf systems [1]

may not affect other customers' hypervisors and guest virtual machines. Ideally, MultiHype achieves the same level of availability and dependability as running each hypervisor on a dedicated MonoHype platform using single physical server.

3.2 Advantages of MultiHype

Comparing with the existing Mono-hypervisor design, a MultiHype system has the following advantages,

First, MultiHype is well-positioned to be the virtualization platform for emerging large scale highly distributed platform. Future multi-processor multi-core sever platform that is armed with multiple memory controllers and multiple many-core processors (e.g., 32 independent cores per processor, and 128 cores per quad-processor platform in 2014 according to AMD) will behave more as a distributed system instead of a monolithic system. Such platform requires a scalable, decentralized hypervisor design to achieve its full potential. In a MonoHype system, all the virtual machine guests share the same hypervisor for handling page faults, exceptions, interrupts, and resource management. The monolithic hypervisor contains numerous mutexes, spinlocks, shared memory states, etc that hinder performance. Running multiple hypervisors with separated states is one of the solutions to address this challenge.

Second, MultiHype enhances cloud security and dependability by eliminating the reliance on shared hypervisor. In multi-tenancy cloud environment, a monolithic hypervisor exposed to virtual machine guests of different customers becomes a potential single-point-of-failure and is attractive to malicious attackers. MultiHype reduces the attack surface associated with shared monolithic hypervisor. In MultiHype, when a single server platform is shared among customers, a different hypervisor can be created for each customer. Virtual machine guests of the same customer are supported by separated hypervisor and, as a result hypervisor vulnerabilities are confined within each customer's own domain.

Table 1 compares three main concepts of cloud computing oriented virtualization environment, single hypervisor over single physical platform, multiple hypervisors over single physical platform, and single hypervisor over multiple platforms.

3.3 Hypervisor/Virtual Machine Realms

Figure 1 shows the concept of MultiHype platform. On a MultiHype server, a VM realm or hypervisor realm refers to all guest virtual machines supported by one VMM or hypervisor. A MultiHype server may constitute multiple concurrent virtual machine realms, and launch new VMMs in on-demand manner. A MultiHype server has a single control plane that administrates the physical machine and retains selective control of resources, including processor cores, physical memory, interrupt assignment, and I/O devices. The control plane can be a virtual machine realm, aka manager realm, while others are regular realms. The manager realm does not run code from guest virtual machine for cloud customers; instead, it allocates physical resources to, bootstrap, and terminate a VMM. After being started, a VMM can function as a normal hypervisor and run independently, i.e., it can manage a number of guest virtual machines and act as a host for the guests. The control plane runs at the highest privilege level, higher than hypervisor's privilege level, therefore ensures isolation among VMMs by allocating or partitioning resources among them. The allocated resource can be physical or virtual. Within each hypervisor, virtual machine guests can be created with additional levels of privileges. For example, within each hypervisor, there can one or multiple administrative guests just like MonoHype system.

For strong isolation purpose, each realm comprises at least one physical processor core and allocated physical RAM space. There is no overlap on processor cores and RAM space for different realms. For a regular realm, its processor cores (one or more) run at lower privilege level than the manager realm. This prevents a regular hypervisor from changing resource allocation made by the control plane. After a hypervisor is started, the control plane delegates control of the allocated physical resources to the started hypervisor. In turn, the hypervisor can further create guest virtual machines and allocate assigned resources by the control plane to the guests.

When a hypervisor starts, it boots from a modified BIOS that bypasses physical RAM initialization. The control plane retains the control of certain physical resources such as physical memory allocation and I/O device discovery. Interrupts for each VM realm are routed and handled by the corresponding hypervisor for the realm. Page faults and exceptions caused by guest virtual machines of a realm are handled by the realm's hypervisor just like in normal MonoHype systems.

The whole system behaves like a distributed system with multiple concurrent virtual machine realms, each having its own hypervisor. Each hypervisor supports context switch between virtual machine mode and hypervisor mode using vm_enter and vm_exit. There is no such context switch between the management realm and a regular realm. The control plane and customer's hypervisor run concurrently using different cores. They communicate with one another through inter-core interrupts and messages. This makes MultiHype fundamentally different from other approaches such as supporting virtual machine creation inside a virtual machine.

3.4 Memory Mapping for Hypervisor/Virtual Machine Realms

To support strong memory partitioning between multiple hypervisors on a single platform, we propose a physical memory remapping mechanism. In particular, physical memory space is divided into chunks of equal size (e.g., 8MB). The control plane assigns physical memory chunks to each hypervisor realm. The remapping mechanism restricts memory access from a hypervisor realm to pre-assigned physical memory regions. This is achieved by a hardware physical memory remapping logic situated in the memory controller. The remapping logic creates a virtual continuous physical memory space for each hypervisor realm. It is programmed by the privileged control plane. A regular hypervisor running at lower privilege level cannot modify or program the remapping logic. For each memory access, e.g., read or write access from a hypervisor or its guest VMs, the memory remapping logic translates the address of the access request to its corresponding physical memory address. Therefore, it serves as a memory access reference monitor and performs access control based on configurations provided by the control plane. For translating memory addresses, the memory controller uses either a realm-to-memory remapping table managed by the control plane, or a local cache of the remapping table. The remapping table cache is part of the memory controller. It resembles TLB inside MMU and

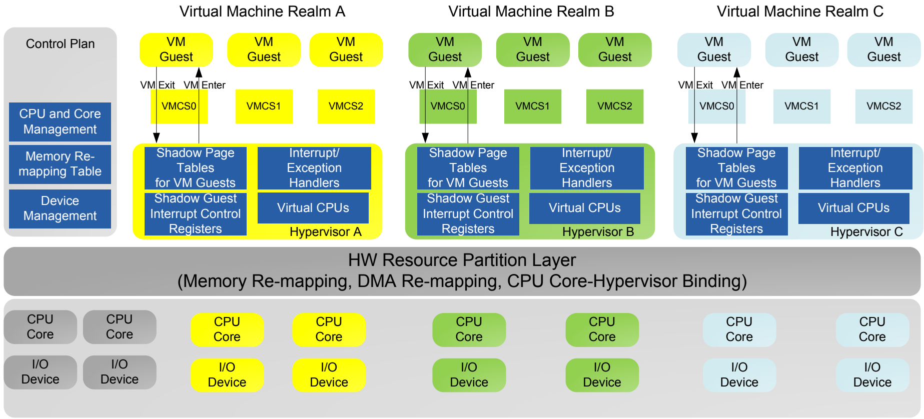


Figure 1: Concept of a MultiHype Platform

caches recently-used entries of the realm-to-physical memory remapping table.

Unlike a regular hypervisor, the control plane or privileged manager realm can access the entire physical memory space without using translation. However, only a portion of the physical memory space is allocated to the control plane. Operating systems or virtual machines within the control plane can use the physical memory space allocated to them freely for their own purposes. The rest physical memory space is reserved for other realms, while the control plane has read/write access rights to it. Overall, physical memory isolation for MultiHype is achieved by restricting memory access from hypervisor realm within the space that is assigned by the control plane, by using the remapping or the address-translation tables.

Figure 2 shows an example physical memory allocation for three VM realms: A, B, and C. When a hypervisor or a VM tries to access to a certain memory location, the remapping hardware looks up the address-translation tables for access permission of the realm to the specific location. If the hypervisor or VM realm tries to access outside of the memory range assigned to it by the control plane, the remapping hardware blocks the access and reports a fault to the control plane, which is achieved by raising exception to the processor core running the control plane. The described physical memory remapping is different from traditional virtual memory management in many aspects. Traditional memory paging and MMU are tied with process management, while our physical memory remapping mechanism is used for partitioning physical memory resources among multiple VM realms. It presents a "virtual" continuous physical memory space for each hypervisor.

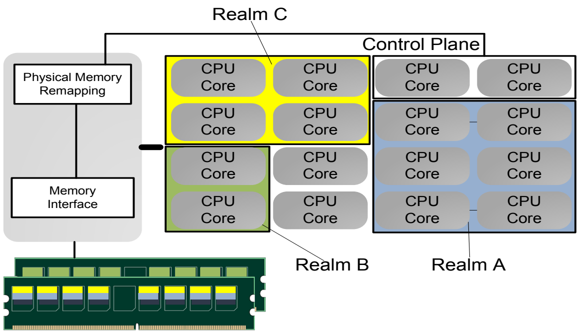


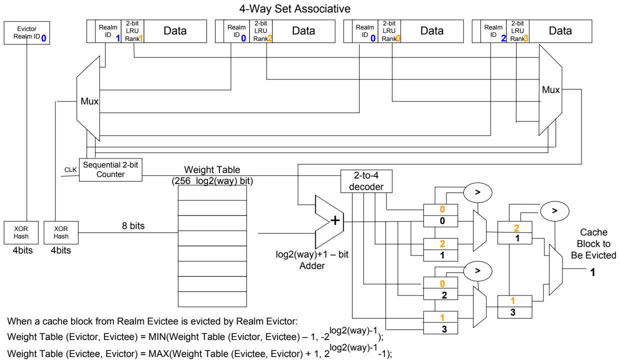
Figure 2: Physical Memory Re-mapping.

3.5 Memory Management

3.5.1 Weighted LRU for Shared L3

Onchip cache is shared by all cores in most today's many-core systems. As a critical hardware resource for achieving high performance, onchip cache becomes an attractive attacking point in multi-tenant computing environment. To address this issue, we design *weighted LRU* in MultiHype. The main idea is to manipulate LRU priorities in order to defend against resource exhaustion exploit of shared cache, which suits for large shared multi-way set associative cache.

Figure 3 provides an illustration of weighted LRU. For each cache block, in addition to tag, valid bit, and other necessary structures, there is a *realm ID* and a LRU rank associated with it. For an 8-way cache, its LRU rank has three bits, which provides a rank of all eight cache blocks of a cache set (7 denotes the least recently accessed cache block and 0 denotes the most recently accessed cache block). Each time, when the cache block is accessed, the rank bits are updated using a map that implements LRU. Design of the map logic is relatively straightforward so we skip the details.



When a cache block from Realm Evictee is evicted by Realm Evictor:

Weight Table (Evictor, Evictee) = MIN(Weight Table (Evictor, Evictee) – 1, $-2^{\log2(way)-1}$);

Weight Table (Evictee, Evictor) = MAX(Weight Table (Evictee, Evictor) + 1, $2^{\log2(way)-1}-1$);

Figure 3: Weighted LRU in MultiHype.

The weight table shown in Figure 3 is used during cache block eviction. Consider that for a cache line, one cache block needs to be evicted to accommodate data from virtual machine realm 0 (evictor), and cache block 0 (candidate evictee) stores data of virtual machine realm 1 and has value 1 as LRU rank. For a pair of evictor and candidate evictee (for instance, realm 0 and realm 1), there is a weight value in the weight table, which can be added to the LRU rank. The summation result is a weighted LRU rank. The range of a weight value is from $-2^{log2(way)-1}$ to $2^{log2(way)-1} - 1$. For each block of a set, a weighted LRU rank can be computed based on the weights stored in the weight table. Using a comparison tree shown in Figure 3, the block with the highest weighted LRU rank can be found after $Log2(Way)$ steps. The block that emerges after the final comparison step will be replaced. After the replacement, the weight table is then updated. If a cache block from virtual machine realm 1 is evicted by virtual machine realm 0, the weight table entry corresponding to the realm pair (0, 1) will decrement by 1, and the weight table entry corresponding to the realm pair (1, 0) will increment by 1. The modified weights increase, in future cache replacements, the probability of evicting cache blocks of realm 0 by accesses from realm 1, and decrease the future probability of evicting cache blocks of realm 1 by accesses from realm 0.

Figure 4 shows one example that highlights the difference between LRU and weighted LRU. Assuming a 4-way set associative cache and two virtual machine realms, there are three write accesses from virtual machine realm 0, all mapped to the same cache line. For each way of the cache line, we use a pair of numbers for storing realm ID, and LRU rank. For example, way 0 stores cached data from realm 0 and its LRU rank is 0. Figure 4 also illustrates values of the weight table (2x2 for two virtual machine realms). For the three write accesses, Figure 4 compares the results of LRU vs. weighted LRU.

Weighted LRU has many attractive properties. First, weighted LRU reduces to standard LRU when there is only one virtual machine realm. This means that when a MonoHype based system is installed, weighted LRU behaves as LRU. Second, as a cache replacement policy, weighted LRU does not statically or dynamically change cache configuration. Consequently, it is orthogonal to and can be used in conjunction with other cache sharing techniques that statically or dynamically share cache among multiple virtual machines.

	Cache Line (4-way)				Weight Table			LRU	Weighted LRU
	Way 0	Way 1	Way 2	Way 3					
Initial Value	(0, 0)	(1,1)	(1,2)	(1,3)		0	1		
					0	0	0		
					1	0	0		
Write access from realm0	(0,1)	(1,2)	(1,3)	(0,0)		0	1	Way 3 evicted	Way 3 evicted
					0	0	-1		
					1	1	0		
Write access from ream 0	(0,2)	(1,3)	(0,0)	(0,1)		0	1	Way 2 evicted	Way 2 evicted
					0	0	-2		
					1	1	0		
Write access from realm 0	(0,0)	(1,3)	(0,1)	(0,2)		0	1	Way 1 evicted	Way 0 evicted
					0	0	-2		
					1	1	0		

Figure 4: Weighted LRU Example (4 Way Associative Cache and 2 VM Realms)

In the current design, MultiHype allocates physical cores to separated hypervisor realms. This eliminates possibilities of sharing L2 or L1 cache by multiple virtual machine realms. Therefore, we only need to consider the shared L3 cache. In weighted LRU, it takes multiple cycles to find out the cache block that should be replaced. There is no performance penalty because the delay is hidden by overlapping WLRU operations with memory fetch.

3.5.2 *Per-realm Memory Throughput Management*

Fair queuing is a technique that originally was designed for managing network bandwidth resources. It allows each flow of packets passing through a network device to have a fair share of network resources. Fair queuing has been demonstrated to help defend against network resource exhaustion attacks. The idea of using fair queuing in memory management was initially exploited in [20] under different application scenarios and CMP context.

The risks and threats of resource exhaustion exploit at system platform level are relatively new. The problem is exacerbated by the multi-tenancy nature of cloud computing. In MultiHype, we apply fair queuing to hypervisor realms to ensure that each of them can have its share of memory bandwidth resources in fair manner. Typically, implementing perfect fairness requires $O(log(n))$ to process each request. Deficit Round-Robin is an approximation of fair queuing that only requires $O(1)$ to process each request [26]. It is simple enough to be implemented in hardware. As shown in [26], Deficit Round-Robin can achieve near perfect fairness.

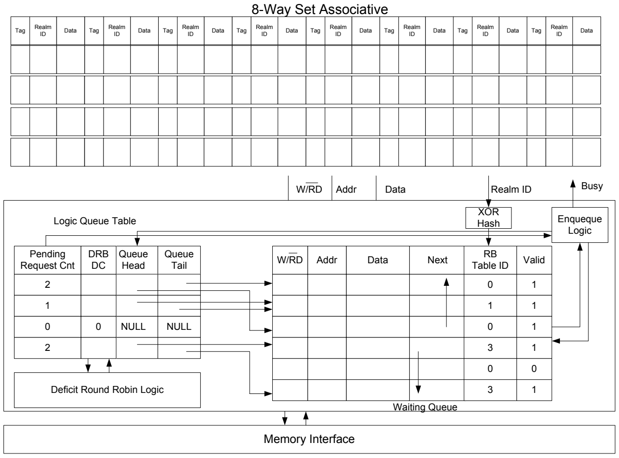


Figure 5: Deficit Round Robin in MultiHype.

Figure 5 illustrates the design of fair queuing in PolyHpye, where each memory access is tagged with its realm ID. There is a logic queue table with fixed number of entries (four in the example). If there are more realms than the queue table entries, an queue entry has to be shared by multiple realms (e.g., XOR the MSB and LSB bits of a realm ID). Each entry has a counter that counts the number of pending memory accesses for that queue, the pointer of queue head, and the pointer of queue tail. Each queue head or tail points to an entry of a memory access waiting queue. All pending memory accesses are buffered in the waiting queue. The waiting queue has only fixed number of entries (e.g., 32 or 64 entries). The waiting queue is shared by all logic queues using head and tail pointers. When the waiting queue is full, upstream logic units with new incoming memory access have to wait or stall until a new waiting queue entry becomes available. Each entry of the logic queue table will be visited by the Deficit Round-Robin Logic in round-robin fashion. For each visited logic queue entry (realm), if there are waiting memory accesses, the requests will be served using deficit Round-Robin, and then the Deficit Round-Robin Logic will visit the next logic queue entry.

3.5.3 *Hardware Cost*

The proposed micro-architectural features including weighted LRU and deficit Round-Robin have only small cost in size. For weighted LRU, each weight needs only three bits (for 8-way cache). The weight table uses only 96B. The largest hardware cost is caused by the realm ID associated with each cache block. Consider a 8-way set associative cache with 4096 sets, and 8 bit realm ID (256 realms), the cost is 32KB. For deficit Round-Robin, the hardware cost is also small. The waiting queue table can have 32 or 64 entries for buffering pending memory accesses. Each entry of the logical queue table has 32 bits. The logic table has variable number of entries with a range from 16 to 256. It costs from 64B to 1KB.

3.6 I/O Support

To virtualize I/O devices, the following operations need to be supported: (1) device discovery and configuration – query I/O devices on a hardware platform and set up the devices' configuration registers for initialization; (2) I/O transactions – transfer data to and from devices including DMA; and (3) interrupts – notify a hypervisor on state updates and events of a device.

I/O virtualization in MonoHype is typically implemented with one of three different approaches: emulation, paravirtualization, and hardware assisted virtualization (e.g., direct assignment on Intel VT-d [2] or single-root IOV where an I/O device can be shared by multiple VMs). Emulation implements I/O devices and hardware in software; paravirtualization requires modification of a guest OS; and hardware assisted I/O virtualization such as VT-d can allocate an I/O device to a VM using IOMMU, a memory address translation for I/O transactions. Single-root IOV enables virtual functions on I/O devices and allows an I/O device to be shared by multiple VMs. For I/O virtualization in MultiHype, we either use I/O proxy or leverage existing hardware assisted I/O virtualization such as device passthrough.

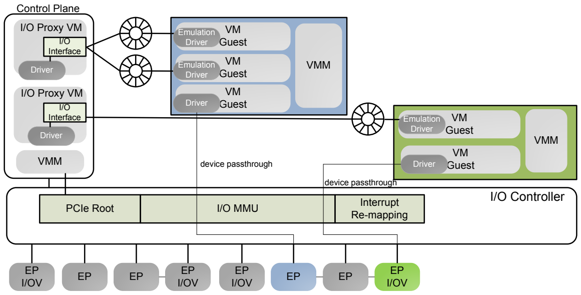


Figure 6: Support I/O Virtualization for MultiHype (I/O Proxy and Device Passthrough

IO Proxy: MultiHype uses device emulation for I/O virtualization, as Figure 6 shows. The control plane consists of a hypervisor and a number of I/O virtual machines, and has the full control of physical I/O resources. I/O devices are allocated to I/O virtual machines in control plane using hardware assisted I/O virtualization such as single-root IOV or VT-d. A regular hypervisor and its VMs can only access the physical I/O resources using the control plane services. In a VM realm, a guest virtual machine uses emulated device drivers, which communicate with the I/O virtual machines in control plane using physical shared memory. This is supported because the control plane has control of the entire physical memory space. Note that the control plane does not run virtual machines for cloud customers. Virtual machines on the control plane are executed by dedicated processor cores in parallel with customers' virtual machines in regular realms. At high level, the system functions like a distributed systems where the control plane and I/O virtual machines act as I/O proxies for the guest virtual machines of a regular realm, by performing I/O transactions and issuing DMA data transfer on behalf of the emulated device drivers. A drawback of this approach is that it does not scale well if there are numerous I/O transactions from multiple guests in a VM realm or VM realms since the control plane is one central place to process the transactions.

Using Existing Device Passthrough Support: Another approach is to leverage the existing hardware support for MonoHype I/O virtualization. It works as follows. The control plane retains the control of I/O devices and assigns I/O resources to guest virtual machines of a realm. This allows I/O virtualization on a MultiHype system using existing hardware I/O virtualization support. The control plane performs device discovery, manages the I/O devices, and assigns devices to guest virtual machines of a VM realm. One guest virtual machine can issue DMA data transfer using hardware I/O virtualization such as VT-d or IOMMU without involving the control plane. When it needs to access protected resources (such as I/O configuration and interrupt management), it first exits into the hypervisor of its realm. The hypervisor then sends an interrupt (such as Inter-processor Interrupt (IPI)) to the processor cores running the control plane. The control plane handles the request and returns results to the hypervisor. For each regular VM realm, its hypervisor cannot perform these I/O controlling functions which are reserved for the control plane. It can only forward the requests from its guests to the control plane.

In our experimentation of MultiHype, both approaches were employed. Both approaches work with the existing I/O virtualization support in MonoHype based systems.

3.7 Interrupts

For the delivery of I/O interrupts to an appropriate VM realm, the current architecture can be minimally changed to support the hypervisor ID. In x86 systems, I/O interrupts are delivered using MSI either via I/O APIC or directly to Local APICs in cores. Destination is determined based on physical or logical IDs of Local APICs in a predefined memory address location. To support the interrupt delivery in MultiHype, an addition of only one register (Hypervisor ID register) is required in I/O APIC, Local APIC and PCIe devices as an extension of hardware virtualization support. Currently, MSI or MSI-x in PCIe 3.0 provides a plenty of space for address encoding. The procedure of interrupt generation and delivery is as follows with a NIC example; A realm requests a network packet to the NIC. Its device driver programs the requester realm ID to the device's hypervisor ID register along with other information. Upon receipt of the packet, the NIC sends an MSI message of which address field embeds the hypervisor ID (realm ID). The MSI message can be broadcasted to the entire realms or directed to the target realm if routers and/or switches in interconnection network inside many-core support the interrupt routing capability. Even simple broadcasting would not incur significant overhead in the interconnection bandwidth considering the intermittent nature of interrupts. The Local APICs compare their hypervisor IDs with the MSI message, and the target (requested) realm takes the interrupt.

4. IMPLEMENTATION & EVALUATION

4.1 Setup and Implementation

We use Bochs [12] – a full-system x86 emulator, and TAXI [29] – a compatible cycle based x86 architecture simulator, to evaluate MultiHype. Bochs models an entire platform including network device, hard drive, VGA, and other devices to support the execution of a complete OS and its applications. In addition, Bochs supports emulation of Intel VMX hardware support for virtualization. TAXI is a Simplescalar simulator with x86 front-end for our performance analysis. Architectural support for MultiHype and the associated resource management features such as weighted LRU, deficit Round-Robin memory bandwidth management, hypervisor realm memory remapping are implemented in both Bochs and TAXI.

We extended Bochs's VMX support and created an emulated hardware partition layer in Bochs. Our emulation framework emulates a multi-core platform. The framework supports configurable logic hardware partition. Multiple hupervisors can be started and executed on an emulated hardware platform using modified Bochs. The modification includes a thin layer of logic hardware partition. Processor cores are bound with hypervisors; that is, virtual machines supported by the same hypervisor can share processor cores, while different hypervisors and their virtual machines run over different processor cores. Each emulated hardware partition can boot a complete hypervisor (Xen 3.3) and run Ubuntu 8.04 Linux distribution.

Our implementation also includes physical memory management for hypervisor realms in Bochs. The performance simulator is extended to support realm based memory mapping, weighted LRU, and deficit Round-Robin. We also integrated an accurate DRAM model [9] to improve system memory modeling, where bank conflicts, page miss, and row miss are all modeled according to SDRAM specification. The processor parameters are listed in Table 2.

Table 2: Platform Parameters in Simulation

Parameters	Values
Frequency	2.0 GHz
Cores	4
Fetch/Decode width	8
Issue/Commit width	8
L1 I-Cache	DM, 16KB, 32B line
L1 D-Cache	DM, 16KB, 32B line
L2 Cache	4way, unified, 32B line, WB cache 256KB for each core
L1/L2 Latency	1 cycle / 6 cycles (256KB)
L3 Cache	8way, 128B line, WB cache, weighted LRU 4MB shared
L3 Latency	16 cycles 4MB
I-TLB	4-way, 128 entries
D-TLB	4-way, 256 entries
Weighted LRU	16 x 3bits table
Deficit Round Robin	16 queues, shared 64 entries
Memory Bus	200MHz, 8B wide
Memory Latency	X-5-5-5 (core clocks), X depends on page status
CAS latency	20 mem bus clocks
Precharge latency (RP)	7 mem bus clocks
RAS-to-CAS (RCD) latency	7 mem bus clocks

We use eight popular open source applications for our evaluation: ffmpeg (a complete cross-platform application to record, convert, and stream audio and video), bzip2 (a popular open-source data compressor), povray (a cross-platform ray tracer), gcc (free compiler for GNU system), pybench (a benchmark suite for python scripting language), octave (a high-level interpreted language for numerical computations, a clone of commercial Matlab), hmmer (an application for searching gene sequence databases), and xalan (an XSLT processor for transforming XML documents into HTML, text, or other XML document types). All applications are installed on Ubuntu 8.04 virtual machine guests and executed together with its host system.

4.2 Hypervisor Subversion Tests

The effectiveness of using multiple hypervisors to defend against attacking on MultiHype platform is evaluated using stress tests. We follow the approach in [22] where hypervisor vulnerability is evaluated by conducting several stress tests on main stream hypervisors. Since most revealed vulnerabilities have been patched, we use multiple older versions of hypervisors in our tests. Our experiments show that hypervisor failures induced by stress tests do not spread to other hypervisors. This confirms the validity of using MultiHype to fend off hypervisor based attacks. In addition, we also try to artificially inject faults into a hypervisor through Bochs. Boches emulates all executed x86 instructions. By altering instruction execution, we inject faults into the hypervisor running ontop. We have observed similar results with this approach. During the test we need to restart the failed hypervisor frequently. However, the fault did not spread to other hypervisors.

4.3 Denial of Resources Tests

We evaluate the strength of MultiHype against denial of resource attacks from co-located malicious virtual machines. We use a simple memory throughput exhaustion application as malicious denial of memory resource exploit. The application runs an infinite loop that walks through a large memory region (several times larger than L3 cache size) and tries to utilize maximum memory throughput by moving data around. The malicious application is executed inside a co-located virtual machine.

We test three collocation scenarios. The the first scenario (dual-hypervisor) includes two concurrently running hypervisors, each runs one guest virtual machine. One guest acts as attacker and is configured to execute the memory throughput exhaustion application. The other guest runs one of the eight benchmark applications. The second scenario (quad-hypervisor) include four concurrently running hypervisors, each runs one guest virtual machine. One guest acts as attacker and is configured to execute the memory throughput exhaustion application. The other three guests in different hypervisors run three of the eight benchmark applications. In the first quad-hypervisor setting (setting one), there are three guests of bzip2, hmmer, ffmpeg, one guest per hypervisor. In the second quad-hypervisor setting (setting two), there are three guests of xalan, gcc, and povray, one per hypervisor. The third scenario (quad dual-guest hypervisors) includes four hypervisors, each has two guests. One guest acts as attacker and is configured to execute the memory throughput exhaustion application. The rest guests are assigned to run bzip2, hmmer, ffmpeg, xalan, gcc, and povray.

5. PERFORMANCE ANALYSIS

Our quantitative performance study focuses on weighted LRU as a cache replacement policy for shared L3 cache, and deficit Round-Robin for memory access management. We evaluate how efficient they prevent memory exhaustion attacks under MultiHype architecture.

5.1 Weighted LRU

From our test results, we found that weighted LRU has three major effects defending against resource exhaustion attacks. First, it promotes fair sharing of cache resources. When there is malicious exploit on cache resources, weighted LRU can boost the amount of cache resources available to the legitimate realms. Second, compared with standard LRU, weighted LRU can significantly increase cache hit rates for guest realms under cache resource exhaustion attack. Third, weighted LRU reduces the likelihood that cached data from legitimate realms are evicted by bombarding cache access requests from resource exhaustion attacks.

Figure 7 shows percentage of L3 cache blocks occupied by legitimate realms and attack realms in the dual realm scenario. As shown in the figure, under the standard LRU, overwhelming amount of cache blocks is occupied by the attack realms. A legitimate realm only occupies insignificant amount of cache blocks – less than 5% in average. In contrast, under weighted LRU, for all the legitimate realms, their shares of L3 cache blocks increase to 12.5% in average. Similar effect of increased L3 occupancy is observed consistently in the other two test scenarios (Figure 9, 10, and 11).

With increased cache residence, the number of L3 cache misses decreases for the legitimate realms. Figure 8 shows L3 cache miss rates for all the tested realms. Compared with standard LRU, weighted LRU reduces L3 cache miss rate for all the legitimate realms from 60% to 41% in average. This effect can be found in the other two test scenarios as well (cf. Figure 9, 10, and 11).

Interestingly, as indicated by Figure 8, cache miss rate remains roughly the same under weighted LRU and standard LRU for the malicious realm. Though the malicious realm occupies less number of cache blocks under weighted LRU, its cache hit rate does not change much. The same effect is observed in all tested scenarios (see Figure 9, 10, and 11). This indicates that in terms of cache hit rate, weighted LRU does not improves cache performance for the legitimate realms at the expense of the attack realm.

Another effect of weighted LRU is that it increases the fairness in cache replacement across realms. Using weighted LRU, a malicious realm has less likelihood unilaterally evicting cache blocks of other hypervisor realms. In certain sense, weighted LRU bestows the legitimate realms power to resist cache evictions by the attack realm. The effect can be observed in Figure 12. When there are multiple realms (the third test scenario, four dual-guest hypervisors), under weighted LRU, cache replacements are more evenly distributed across the realms that have light memory access demand. The effect can be found in Figure 13.

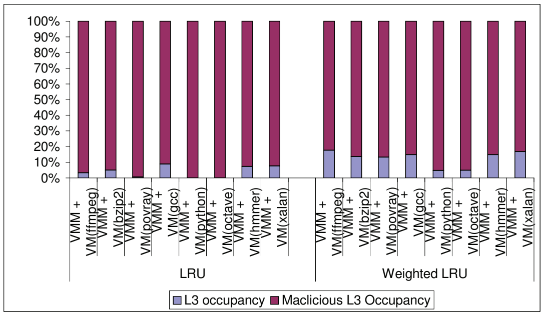


Figure 7: L3 Occupancy: One Legitimate Realm vs. One Malicious Realm.

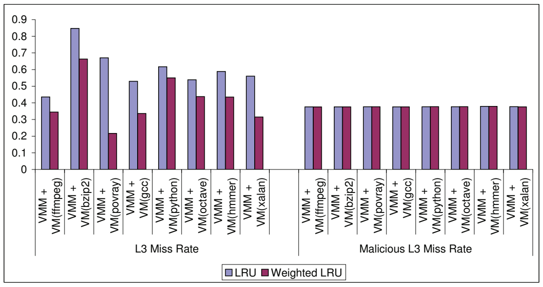


Figure 8: L3 Miss Rate: One Legitimate Realm vs. One Malicious Realm.

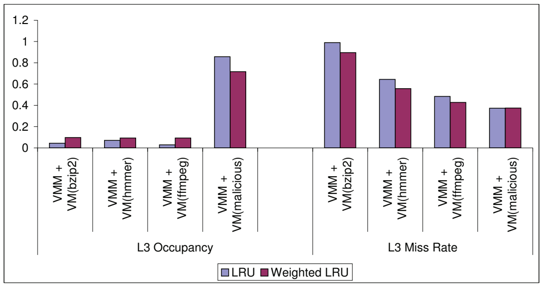


Figure 9: L3 Occupancy and L3 Miss Rate: Setting One of Four Realms.

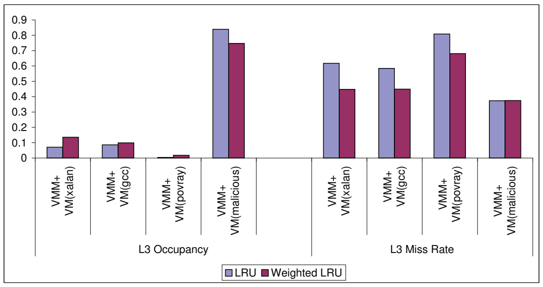


Figure 10: L3 Occupancy and L3 Miss Rate: Setting Two of Four Realms.

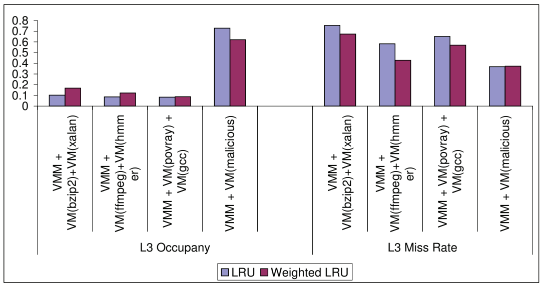


Figure 11: L3 Occupancy and L3 Miss Rate: Four Dual-Guest Realms.

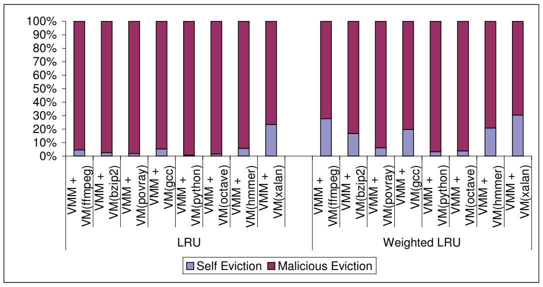


Figure 12: L3 Eviction Profile: One Legitimate Realm vs. One Malicious Realm.

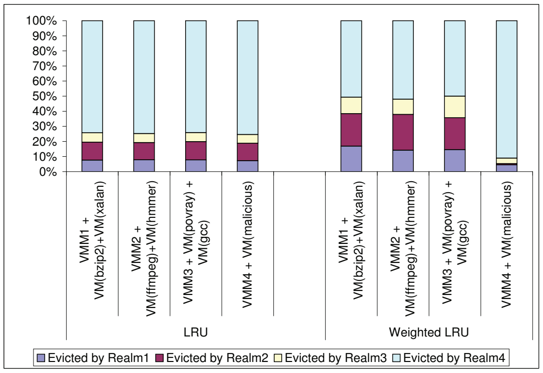


Figure 13: L3 Eviction Profile: Four Dual-Guest Realms.

5.2 Memory Utilization

Our next defense line against memory exhaustion attacks is at the memory interface using deficit Round-Robin. As aforementioned, deficit Round-Robin is a flavor of fair queueing, with the main advantage of simplicity over other fair queueing implementations. Deficit Round-Robin is easy to be implemented in hardware. In our test, we compare the memory utilization between deficit Round Robin and FIFO with a memory exhaustion attack application, which generates overwhelming amount of memory accesses. When FIFO is used for handling memory access requests, requests from the legitimate realms suffer because they have to wait for all outstanding memory requests from the malicious application to complete. Deficit Round- Robin iterates over all the memory request waiting queues and guarantees memory bandwidth allocation to all realms.

For each memory access, we measure its waiting time (time spent in the memory request queue or FIFO before memory bandwidth is allocated to the request). We calculate the average waiting time for all memory accesses under FIFO and deficit Round Robin. For each realm (legitimate and malicious), we compare the per-realm average waiting time against the overall waiting time across all realms. As shown in Figure 14, memory requests from the legitimate realms have much less relative waiting time under deficit Round Robin than that with the relative waiting time under FIFO. With deficit Round Robin, memory requests from the legitimate realms only need to wait 49% of the overall average waiting time. In contrast, with FIFO, memory requests from the legitimate realms need to wait 67% of the overall average waiting time. For the attacking realm, in both cases (FIFO and deficit Round Robin), the average waiting time is roughly the same as the overall average waiting time or slightly higher. This effect can be found in all other three test scenarios (see Figure 15 and 16). Our quantitative evaluation results show that both weighted LRU and deficit Round-Robin are effective against memory resource exhaustion attacks from a malicious realm and guest.

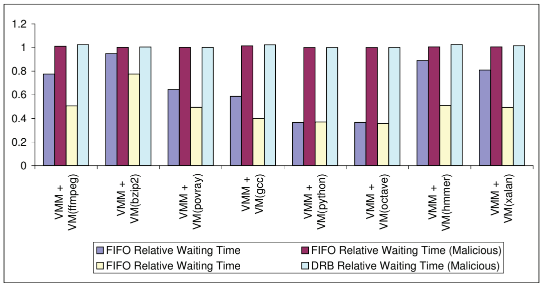


Figure 14: Relative Memory Access Waiting Time: One Legitimate Realm vs. One Malicious Realm.

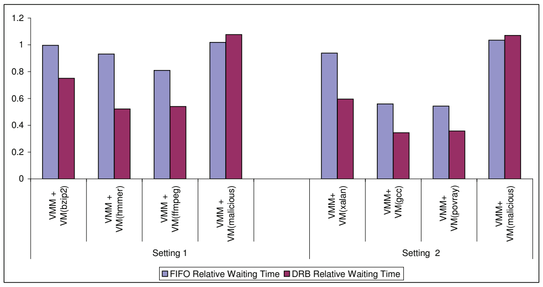


Figure 15: Relative Memory Access Waiting Time: Four Realms.

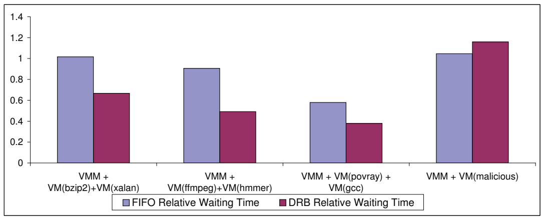


Figure 16: Relative Memory Access Waiting Time: Four Dual-Guest Realms.

6. RELATED WORK

Hardware and architectural support for CPU and I/O virtualization To our best knowledge, majority of the published studies and designs in this space focus on supporting single hypervisor based systems [2, 28]). In contrast to these systems, MultiHype is one of the first that employ new architectural features to support multi-hypervisor based platform. In [5], Ben Yehuda et al. propose a solution for supporting nested virtual machines. Compared with MultiHype, this is a special type of mono hypervisor system as all nested virtual machines depend on a single bottom layer hypervisor. Thus, in many-core based environments, this cannot scale with a great many guest VMs because a single hypervisor should handle them. In addition, this has potential security problems because all guest VMs share the underlying raw hardware. NoHype [14] is an architecture that uses hardware virtualization extensions to remove any need for virtualization layer such as hypervisor. The architecture includes running one virtual machine per core, hardware enforced memory partitioning, dedicated (virtual) devices. However, NoHype cannot adjust resources partition between virtual machines dynamically because each guest virtual machine controls hardware directly.

Resource management in a multicore processor Many researchers in the last few years have explored the resource sharing problems of a multicore processor. These problems include fairness, QoS, or even DoS vulnerability in a shared cache [21, 6], memory bandwidth [20, 18], or both of them concurrently [31]. In contrast to the related studies that dynamically allocate cache and memory resources, MultiHype focuses on strong isolation between concurrently executed hypervisor realms over a single physical platform. Unlike related work that dynamically allocate cache and memory resources based on demand, our system does not give more cache and/or memory resources to any malicious application. The described weighted LRU cache replacement policy does not explicitly or directly manage cache resources (e.g., space and bandwidth); instead, it defends against malicious cache exploits indirectly by manipulating LRU weights of a shared cache. In a mono hypervisor environment, the policy reduces to standard LRU. Different from the related work, we use deficit Round-Robin to manage memory resources for each hypervisor realm, which introduces low cost and hardware efficient approximation to fair queuing, and is suited for independency of individual hypervisor realms. Furthermore, defending against resource exhaustion exploit is only one aspect of the strong isolation. Other aspect includes confining any fault or failure that occurs within a hypervisor realm.

Micro-kernel based hypervisor NOVA [27] is a micro-kernel based hypervisor that uses a thin and simple virtualization layer to reduce the attack surface and as a result improve system security. MultiHype is orthogonal and complementary to this approach. Comparing with this, MultiHype eliminates the necessity of sharing hypervisor among different cloud customers on a single platform and thereby improves platform reliability, scalability, and security.

Multi-kernel support Baumann et al. [4] proposed a new operating system, Barrelfish multikernel, which focuses on the scalability of heterogeneous multicore systems. Barrelfish treats each core as an independent entity as if each core is a node in a distributed system. In such a system, cores communicate using messages and do not share memory. In contrast, MultiHype intends to scale in a many-core platform running multiple hypervisors on a single physical server. Unlike Barrelfish, each hypervisor can manage multiple cores while running independently.

7. CONCLUSION

We present the design and evaluation of MultiHype, a platform architecture to support multiple hypervisors on a single physical platform by leveraging the emerging many-core based cloud-on-chip processor. Each hypervisor in MultiHype manages guest virtual machines like traditional virtualized platform. The strong isolation between hypervisors and their realms is achieved by the separation of physical resources provided by a control plane of the platform, which includes a new LRU based cache replacement policy for preventing cache exhaustion attacks, and efficient memory management approach for defending against denial of resource attacks on physical memory in multi-tenancy cloud environment. These micro-architectural features confines the vulnerabilities of one hypervisor or its virtual machine within its own domain, which makes the MultiHype platform more resilient to malicious attacks and failures in cloud computing environment. Our evaluations using Bochs emulator and cycle based x86 simulation show both qualitatively and quantitatively the effectiveness of MultiHype as a new platform architecture with improved security and dependability beyond legacy virtualization platform.

8. REFERENCES

[1] 3Leaf Systems. Next generation hybrid systems for hpc. http://www.3leafsystems.com/download/3leaf_wt_paper_Next_Gen_Hybrid_Sys\%tems_for_HPC.pdf, 2009.

[2] Abramson, D., Jackson, J., Muthrasanallur, S., Neiger, G., Regnier, G., Sankaran, R., Schoinas, I., Uhlig, R., Vembu, B., and Weigert, J. Intel Virtualization Technology for directed I/O. *Intel Technology Journal 10*, 3 (Aug. 2006), 179–192.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Virtual machine monitors: Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles: the Sagamore, Bolton Landing, Lake George, New York, USA, October 19–22, 2003* (New York, NY 10036, USA, Dec. 2003), ACM, Ed., vol. 37(5) of *Operating systems review*, ACM Press, pp. 164–177. ACM order number 534030.

[4] Baumann, A., Barham, P., Dagand, P., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Sch ”upbach, A., and Singhania, A. The multikernel: a new os architecture for scalable multicore systems. In *SOSP* (2009), vol. 9, Citeseer, pp. 29–44.

[5] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., and Yassour, B.-A. The turtles project: design and implementation of nested virtualization. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 1–6.

[6] Chang, J., and Sohi, G. Cooperative cache partitioning for chip multiprocessors. In *Proceedings of the 21st annual international conference on Supercomputing* (2007), ACM, pp. 242–252.

[7] Ferrie, P. Attacks on virtual machine emulators. *Symantec Security Response 5* (2006).

[8] Frank Gens, Robert P Mahowald, R. L. V. An empirical study into the security exposure to hosts of hostile virtualized environments, 2007.

[9] Gries, M., and Romer., A. Performance Evaluation of Recent DRAM Architectures for Embedded Systems. In *TIK Report Nr. 82, Computing Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich* (November 1999).

[10] Heiser, J., and Nicolett, M. Assessing the security risks of cloud computing. http://www.gartner.com/DisplayDocument?id=685308, 2009.

[11] Held, J., Bautista, J., and Koehl, S. White paper from a few cores to many: A tera-scale computing research review.

[12] K. Lawton. Welcome to the Bochs x86 PC Emulation Software Home Page. http://www.bochs.com.

[13] Karger, P. A., and Safford, D. I/O for virtual machine monitors: Security and performance issues. *IEEE Security & Privacy 6*, 5 (2008), 16–23.

[14] Keller, E., Szefer, J., Rexford, J., and Lee, R. B. Nohype: virtualized cloud infrastructure without the virtualization. In *Proceedings of the 37th annual international symposium on Computer architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 350–361.

[15] King, S. T., Chen, P. M., min Wang, Y., Verbowski, C., Wang, H. J., and Lorch, J. R. Subvirt: Implementing malware with virtual machines. In *IEEE Symposium on Security and Privacy* (2006), pp. 314–327.

[16] Kortchinsky, K. Cloudburst – hacking 3D and breaking out of VMware. In *Black Hat USA* (2009).

[17] Mell, P. Nist presentation on effectively and securely using the cloud computing paradigm v26. http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2009.

[18] Moscibroda, T., and Mutlu, O. Memory performance attacks: Denial of memory service in multi-core systems. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (2007), USENIX Association, p. 18.

[19] Neiger, G., Santoni, A., Leung, F., Rodgers, D., and Uhlig, R. Intel Virtualization Technology: Hardware support for efficient processor virtualization. *Intel Technology Journal 10*, 3 (Aug. 2006), 167–177.

[20] Nesbit, K. J., Aggarwal, N., Laudon, J., and Smith, J. E. Fair queuing memory systems. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (Washington, DC, USA, 2006), MICRO 39, IEEE Computer Society, pp. 208–222.

[21] Nesbit, K. J., Laudon, J., and Smith, J. E. Virtual private caches. In *Proceedings of the 34th annual international symposium on Computer architecture* (New York, NY, USA, 2007), ISCA '07, ACM, pp. 57–68.

[22] Ormandy, T. An empirical study into the security exposure to hosts of hostile virtualized environments. In *CanSecWest* (2007).

[23] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. 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* (New York, NY, USA, 2009), CCS '09, ACM, pp. 199–212.

[24] Rutkowska, J. Blue pill. In *Black Hat USA* (2006).

[25] Secunia. Advisory sa37081 - VMware ESX sever uodate for DHCP, kernel, and JRE. http://secunia.com/advisories/37081/.

[26] Shreedhar, M., and Varghese, G. Efficient fair queuing using deficit round robin. *IEEE Trans. Net* (1996).

[27] Steinberg, U., and Kauer, B. NOVA: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European conference on Computer systems* (New York, NY, USA, 2010), EuroSys '10, ACM, pp. 209–222.

[28] Uhlig, R. Forward: Intel Virtualization Technology: Taking virtualization mainstream on Intel architecture platforms. *Intel Technology Journal 10*, 3 (Aug. 2006), v–vi.

[29] Vlaovic, S., and Davidson, E. S. TAXI: Trace Analysis for X86 Interpretation. In *Proceedings of the 2002 IEEE International Conference on Computer Design* (2002).

[30] Wojtczuk, R. Subverting the Xen hypervisor. In *Black Hat USA* (2008).

[31] Woo, D. H., and Lee, H.-H. Analyzing performance vulnerability due to resource denial of service attack on chip multiprocessors. In *Workshop on Chip Multiprocessor Memory Systems and Interconnects* (2007).