Multi-Hypervisor Virtual Machines: Enabling An Ecosystem of Hypervisor-level Services

Kartik Gopalan, Rohith Kugve, Hardik Bagdi, Yaohui Hu – Binghamton University
Dan Williams, Nilton Bila – IBM T.J. Watson Research Center

Funded by NSF
Hypervisors

- A thin and secure layer in the cloud
  -- or --
Hypervisors

- A thin and secure layer in the cloud
  -- or --
- Feature-filled cloud differentiators
  - Migration
  - Checkpointing
  - High availability
  - Live Guest Patching
  - Network monitoring
  - Intrusion detection
  - Other VMI
Lots of third-party interest in hypervisor-level services

- Ravello
- Bromium
- XenBlanket
- McCafe DeepDefender
- Secvisor
- Cloudvisor
- And more...

- But limited support for third party services from base hypervisor.
How can a guest use multiple third-party hypervisor-level services?

• Our Solution: **Span virtualization**

• One guest controlled by multiple coresident hypervisors.
Outline

• Why multi-hypervisor virtual machines?
• Design of Span Virtualization
• Evaluations
• Related Work
• Conclusions and Future Work
Option 1: Fat hypervisor

• All services run at the most privileged level.

• But...hypervisor cannot trust third-party services in privileged mode.
Option 2: Native user space services

- Services run natively in the user space of the hypervisor

- Services control guest indirectly via the hypervisor

- E.g. QEMU with KVM, uDenali

- But...Potentially large user-kernel interface
  - event interposition and system calls

*Cloud providers reluctant to run third-party services natively, even if in user space.*
Option 3: Service VMs

- Run services inside deprivileged VMs

- Services control guest indirectly via hypercalls and events

- **Single trusted Service VM**
  - Runs all services
  - E.g. Domain0 in Xen

---

Service VM

Hypervisor

Guest 1

Service A  
Service B  
Service C
Option 3: Service VMs

• Run services inside deprivileged VMs

• Services control guest indirectly via hypercalls and events

• **Multiple service VMs**
  • One per service
  • Deprivileged and restartable
  • E.g. Service Domains in Xoar
Option 3: Service VMs

- Run services inside deprivileged VMs

- Services control guest indirectly via hypercalls and events

- **Multiple service VMs**
  - One per service
  - Deprivileged and restartable
  - E.g. Service Domains in Xoar

*Lack direct control over ISA-level guest state*
- Memory mappings, VCPU scheduling, port-mapped I/O, etc.
Option 4: Nested Virtualization

• Services run in a depriviledged L1 hypervisor, which runs on L0.

• Services control guest at virtualized ISA level.

• But ... multiple services must reside in the same L1, i.e. fat L1.

• Vertically Stack L1 hypervisors?
  - More than two levels of nesting is inefficient.
Our solution: *Span Virtualization*

- Allow multiple coresident L1s to concurrently control a common guest
  - i.e. Horizontal layering of L1 hypervisors

- Guest is a *multi-hypervisor virtual machine*

- Each L1
  - Offers guest services that augment L0’s services.
  - Controls one or more guest resources
Design Goals of Span Virtualization

• **Guest Transparency**
  • Guest remains unmodified

• **Service Isolation**
  • L1s controlling the same guest are unaware of each other.
Guest Control operations

- L0 supervises which L1 controls which Guest resource
  - Memory, VCPU and I/O

- L0 and L1s communicate via Traps/Faults (implicit) and Messages (explicit)

- Operations:
  - **Attach** an L1 to a specified guest resource
  - **Detach** an L1 from a guest resource
  - **Subscribe** an attached L1 to receive guest events (currently memory events)
  - **Unsubscribe** an L1 from a subscribed guest event
Control over Guest Resources

• **Guest Memory**
  • Shared: All hypervisors have the same consistent view of guest memory

• **Guest VCPUs**
  • Exclusive: All guest VCPUs are controlled by one hypervisor at any instant

• **Guest I/O devices**
  • Exclusive: Different virtual I/O devices of a guest may be controlled by different hypervisors

• **Control Transfer**
  • Control over guest VCPUs and I/O devices can be transferred from one L1 to another via L0.
Memory Translation

**Single-Level Virtualization**

- **VA** → **Page Table** → **GPA** → **EPT** → **HPA**

- **Nested virtualization**
  - L0 is responsible for the traditional paging hardware, which maps virtual to physical addresses.
  - The guest is granted control over its own memory space.

- **Continuous vs. Transient Control**
  - Continuous control means that an L1 requires control over the guest's resources continuously, with no interruption.
  - Transient control means that an L1 can provide short service to a guest and then release control back to the hypervisor controlling the guest.

- **Isolation and Communication**
  - L0 and L1s must be isolated from each other, with L1s only able to communicate with L0.
  - L1s must remain isolated from each other during execution.

- **Event Processing**
  - Messages allow an L1 to directly request guest control from L0.
  - Transient control means that an L1 can provide a short service to a guest and then release control back to the hypervisor controlling the guest.

- **Channel**
  - A bidirectional processing module relays guest memory faults that need to be handled by L1.

- **Performance**
  - L1s are isolated from each other, which improves performance.
  - L1s must remain isolated from each other during execution.

- **EPT**
  - Extended Page Tables (EPT) are used to map guest pages to pages in the physical address space.

- **Virtual EPT**
  - The hypervisor uses an EPT to map guest virtual addresses to guest physical addresses (VA to GPA in the figure).

- **Nested virtualization**
  - L0 maintains EPT to map guest pages to pages in the physical address space.
  - L1s maintain a shadow EPT to map their guest physical addresses to guest physical address space.

- **Nested virtualization with Shadow EPT**
  - L1s maintain a shadow EPT that maps their guest physical addresses to guest physical address space, allowing them to maintain their own mappings while being isolated from the hypervisor.

- **Continuous control**
  - Continuous control means that an L1 requires control over the guest's resources continuously, with no interruption.

- **Transient control**
  - Transient control means that an L1 can provide a short service to a guest and then release control back to the hypervisor controlling the guest.

- **Interrupts**
  - Some explicit messages, such as guest I/O requests, could be replaced with interrupts.

- **Guest I/O requests**
  - Guest I/O requests can be handled by L1 to provide continuous control over guest resources.

- **Guest OS**
  - A guest OS can use a single-level virtualization feature called Span virtualization, which allows the guest to maintain the mappings between its virtual and physical address spaces.

- **HPA**
  - Host Physical Address.

- **Virtual EPT**
  - The guest is similarly granted continuous control over its memory, without trapping into the hypervisor.

- **IOMMU**
  - The traditional paging hardware is used to map virtual addresses to physical addresses (VA to HPA).

- **Guest permissions**
  - Guest memory permissions are controlled by the combination of permissions in the guest page table and the EPT.

- **Guest memory faults**
  - Whenever the guest attempts to access a page that is either not present or protected in the EPT, the hardware generates an EPT fault and traps into the hypervisor.

- **HPA**
  - Host Physical Address.

- **Virtual EPT**
  - The hypervisor uses an EPT to map guest virtual addresses to guest physical addresses (VA to GPA in the figure).

- **Nested virtualization**
  - For nested virtualization, the guest is similarly granted control over its own memory space.

- **EPT**
  - Extended Page Tables (EPT) are used to map guest pages to pages in the physical address space.

- **Virtual EPT**
  - The hypervisor uses an EPT to map guest virtual addresses to guest physical addresses (VA to GPA in the figure).

- **Continuous control**
  - Continuous control means that an L1 requires control over the guest's resources continuously, with no interruption.

- **Transient control**
  - Transient control means that an L1 can provide a short service to a guest and then release control back to the hypervisor controlling the guest.

- **Interrupts**
  - Some explicit messages, such as guest I/O requests, could be replaced with interrupts.

- **Guest I/O requests**
  - Guest I/O requests can be handled by L1 to provide continuous control over guest resources.

- **Guest OS**
  - A guest OS can use a single-level virtualization feature called Span virtualization, which allows the guest to maintain the mappings between its virtual and physical address spaces.

- **HPA**
  - Host Physical Address.

- **Virtual EPT**
  - The hypervisor uses an EPT to map guest virtual addresses to guest physical addresses (VA to GPA in the figure).

- **Nested virtualization**
  - For nested virtualization, the guest is similarly granted control over its own memory space.

- **EPT**
  - Extended Page Tables (EPT) are used to map guest pages to pages in the physical address space.

- **Virtual EPT**
  - The hypervisor uses an EPT to map guest virtual addresses to guest physical addresses (VA to GPA in the figure).

- **Continuous control**
  - Continuous control means that an L1 requires control over the guest's resources continuously, with no interruption.

- **Transient control**
  - Transient control means that an L1 can provide a short service to a guest and then release control back to the hypervisor controlling the guest.

- **Interrupts**
  - Some explicit messages, such as guest I/O requests, could be replaced with interrupts.

- **Guest I/O requests**
  - Guest I/O requests can be handled by L1 to provide continuous control over guest resources.

- **Guest OS**
  - A guest OS can use a single-level virtualization feature called Span virtualization, which allows the guest to maintain the mappings between its virtual and physical address spaces.

- **HPA**
  - Host Physical Address.

- **Virtual EPT**
  - The hypervisor uses an EPT to map guest virtual addresses to guest physical addresses (VA to GPA in the figure).

- **Continuous control**
  - Continuous control means that an L1 requires control over the guest's resources continuously, with no interruption.

- **Transient control**
  - Transient control means that an L1 can provide a short service to a guest and then release control back to the hypervisor controlling the guest.

- **Interrupts**
  - Some explicit messages, such as guest I/O requests, could be replaced with interrupts.

- **Guest I/O requests**
  - Guest I/O requests can be handled by L1 to provide continuous control over guest resources.

- **Guest OS**
  - A guest OS can use a single-level virtualization feature called Span virtualization, which allows the guest to maintain the mappings between its virtual and physical address spaces.

- **HPA**
  - Host Physical Address.
Memory Translation

Single-Level Virtualization

Nested Virtualization

[Diagram showing memory translation processes for single-level and nested virtualization]
Memory Translation

**Single-Level Virtualization**

\[ \text{VA} \rightarrow \text{Page Table} \rightarrow \text{GPA} \rightarrow \text{EPT} \rightarrow \text{HPA} \]

**Nested Virtualization**

\[ \text{VA} \rightarrow \text{Page Table} \rightarrow \text{GPA} \rightarrow \text{Virtual EPT} \rightarrow \text{L1PA} \rightarrow \text{EPT}_{L1} \rightarrow \text{HPA} \]

**Span Virtualization**

\[ \text{VA} \rightarrow \text{Page Table} \rightarrow \text{GPA} \rightarrow \text{Virtual EPT} \rightarrow \text{L1PA} \rightarrow \text{EPT}_{L1} \rightarrow \text{EPT}_{Guest} \rightarrow \text{HPA} \]
Synchronizing Guest Memory Maps

- Guest physical memory to Host physical memory translation should be the same regardless of the translation path.

- L0 syncs Shadow EPTs and \( EPT_{L1s} \)
  - Guest faults
  - Virtual EPT modifications by L1
  - When L1 directly accesses guest memory

- L1s subscribe to guest memory events via L0
  - E.g. to track write events for dirty page tracking

---

**Diagram Description**

- **Process VA**
- **Page Table**
- **GPA**
- **L0**
- **L1 Hypervisor(s)**
- **Virtual EPT**
- **EPT_{L1}**
- **Shadow EPT**
- **HPA**
- **Span Guest**
- **Memory Event Emulator**
- **Event Notifications**
- **Guest Event Handling**
- **Virtual EPT Trap Handler**
- **Event Subscription Service**
I/O Control

- We consider para-virtual I/O in this work
I/O Control

• We consider para-virtual I/O in this work
• A Span Guest’s I/O device and VCPUs may be controlled by different L1s
VCPU control

• Simple for now.

• All VCPUs controlled by one hypervisor
  • Either by L0 or one of the L1s

• Can we distribute VCPUs among L1s?
  • Possible, but no good reason why.
  • Requires expensive IPI forwarding across L1s
  • Complicates memory synchronization.
Implementation

- **Guest**: Unmodified Ubuntu 15.10, Linux 4.2

- **L0 and L1**
  - QEMU 1.2 and Linux 3.14.2
  - Modified nesting support in KVM/QEMU
  - L0: 980+ lines in KVM and 500+ lines in QEMU
  - L1: 300+ lines in KVM and 380+ in QEMU

- **Guest controller**
  - User space QEMU process
  - Guest initialization, I/O emulation, Control Transfer, Migration, etc

- **I/O: virtio-over-virtio**
  - Direct assignment: future work

- **Message channel**
  - For I/O kick and interrupt forwarding
  - Currently using UDP messages and hypercalls

- **Control Transfer**
  - Guest VCPUs and virtio devices can be transferred between L1s and L0
  - Using attach/detach operations
Example 1: Two L1s controlling one Guest

- **Guest**: Infected with rootkit
- **L1a**: Monitoring network traffic
- **L1b**: Running VMI (Volatility)

---

```
L1a: Network Monitoring
```

```
Guest infected with KBeast

L1b: Volatility
```
Example 2: Guest mirroring

- L1a runs Volatility

- L1b runs Guest Mirroring
  - Periodically copy dirty guest pages
  - Requires subscription on write events

- Guest runs iPerf
  - ~800Mbps when mirrored every 12 seconds. Same as standard nested.
  - ~600Mbps every 1 second.
  - 25% impact with high frequency dirty page tracking
Example 3: Live Hypervisor Replacement

- Replace hypervisor underneath a live Guest
  - L1 runs a full hypervisor
  - L0 acts as a thin switching layer

- Replacement operation
  - Attach new L1
  - Detach old L1

- 740ms replacement latency, including memory co-mapping

- 70ms guest downtime
  - During VCPU and I/O state transfer
Macrobenchmarks

<table>
<thead>
<tr>
<th>L0</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem</td>
<td>CPUs</td>
<td>Mem</td>
</tr>
<tr>
<td>Single</td>
<td>128GB</td>
<td>12</td>
</tr>
<tr>
<td>Nested</td>
<td>128GB</td>
<td>12</td>
</tr>
<tr>
<td>Span0</td>
<td>128GB</td>
<td>12</td>
</tr>
<tr>
<td>Span1</td>
<td>128GB</td>
<td>12</td>
</tr>
</tbody>
</table>

Guest Workloads
- Kernbench: repeatedly compiles the kernel
- Quicksort: repeatedly sorts 400MB data
- iPerf: Measures bandwidth to another host

Hypervisor-level Services
- Network monitoring (tcpdump)
- VMI (Volatility)
Macrobenchmarks

<table>
<thead>
<tr>
<th></th>
<th>L0</th>
<th></th>
<th>L1</th>
<th></th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem</td>
<td>CPUs</td>
<td>Mem</td>
<td>VCPUs</td>
<td>Mem</td>
<td>VCPUs</td>
</tr>
<tr>
<td>Single</td>
<td>128GB</td>
<td>12</td>
<td>3GB</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>Nested</td>
<td>128GB</td>
<td>12</td>
<td>16GB</td>
<td>8</td>
<td>3GB</td>
</tr>
<tr>
<td>Span0</td>
<td>128GB</td>
<td>12</td>
<td>8GB</td>
<td>4</td>
<td>3GB</td>
</tr>
<tr>
<td>Span1</td>
<td>128GB</td>
<td>12</td>
<td>8GB</td>
<td>4</td>
<td>3GB</td>
</tr>
</tbody>
</table>

Guest Workloads

- Kernbench: repeatedly compiles the kernel
- Quicksort: repeatedly sorts 400MB data
- iPerf: Measures bandwidth to another host

Hypervisor-level Services

- Network monitoring (tcpdump)
- VMI (Volatility)

![Figure 8: No-op Mode: Normalized performance when no services run in host, L0, or L1s. The L0 controls the virtio device.](a) Kernbench

![Figure 9: Service Mode: Normalized performance with hypervisor-level services network monitoring and volatility. L0 runs guest VCPU and I/O devices. L1 runs network monitoring and controls the guest's network device; L1b runs volatile L1s with a fresh instance while the guest executes.](b) Quicksort

![Figure 10: Overhead of attaching an L1 to a guest's memory, VCPU, and I/O devices. The time taken to attach memory of a 1GB Span guest is about 8.3 Micro Benchmarks](c) iPerf
Macrobenchmarks

<table>
<thead>
<tr>
<th></th>
<th>L0</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem</td>
<td>CPUs</td>
<td>Mem</td>
</tr>
<tr>
<td>Single</td>
<td>128GB</td>
<td>12</td>
<td>3GB</td>
</tr>
<tr>
<td>Nested</td>
<td>128GB</td>
<td>12</td>
<td>16GB</td>
</tr>
<tr>
<td>Span0</td>
<td>128GB</td>
<td>12</td>
<td>8GB</td>
</tr>
<tr>
<td>Span1</td>
<td>128GB</td>
<td>12</td>
<td>8GB</td>
</tr>
</tbody>
</table>

**Guest Workloads**
- Kernbench: repeatedly compiles the kernel
- Quicksort: repeatedly sorts 400MB data
- iPerf: Measures bandwidth to another host

**Hypervisor-level Services**
- Network monitoring (tcpdump)
- VMI (Volatility)
**Microbenchmarks**

<table>
<thead>
<tr>
<th></th>
<th>Single</th>
<th>Nested</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPT Fault</td>
<td>2.4</td>
<td>2.8</td>
<td>3.3</td>
</tr>
<tr>
<td>Virtual EPT Fault</td>
<td>-</td>
<td>23.3</td>
<td>24.1</td>
</tr>
<tr>
<td>Shadow EPT Fault</td>
<td>-</td>
<td>3.7</td>
<td>4.1</td>
</tr>
<tr>
<td>Message Channel</td>
<td>-</td>
<td>-</td>
<td>53</td>
</tr>
<tr>
<td>Memory Event Notify</td>
<td>-</td>
<td>-</td>
<td>103.5</td>
</tr>
</tbody>
</table>

Low-level latencies in Span virtualization
Related Work

- User space Services
  - Microkernels, library OS, uDenali, KVM/QEMU, NOVA

- Service VMs
  - Dom0 in Xen, Xoar, Self-Service Cloud

- Nested virtualization
  - Belpaire & Hsu, Ford et. al, Graf & Roedel, Turtles
  - Ravello, XenBlanket, Bromium, DeepDefender, Dichotomy

- Span virtualization is the first to address multiple third-party hypervisor-level services to a common guest
Summary: Span Virtualization

• We introduced the concept of a *multi-hypervisor virtual machine*  
  • that can be concurrently controlled by multiple coresident hypervisors

• Another tool in a cloud provider’s toolbox  
  • to offer compartmentalized guest-facing third-party services

• Future work  
  • Faster event notification and processing  
  • Direct device assignment to L1s or Guest  
  • Possible to support unmodified L1s?  
    • Requires L1s to support partial guest control. Current L1s assume full control.

• Code to be released after porting to newer KVM/QEMU
Questions?
Backup slides
Comparison

<table>
<thead>
<tr>
<th></th>
<th>Level of Guest Control</th>
<th>Impact of Service Failure</th>
<th>Additional Performance Overheads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Virtualized ISA</td>
<td>Partial or Full</td>
<td>L0</td>
</tr>
<tr>
<td>Single-level</td>
<td>Yes</td>
<td>Full</td>
<td>Fails</td>
</tr>
<tr>
<td>User space</td>
<td>No</td>
<td>Partial</td>
<td>Protected</td>
</tr>
<tr>
<td>Service VM</td>
<td>No</td>
<td>Partial</td>
<td>Partial</td>
</tr>
<tr>
<td>Nested</td>
<td>Yes</td>
<td>Full</td>
<td>Protected</td>
</tr>
<tr>
<td>Span</td>
<td>Yes</td>
<td>Both</td>
<td>Protected</td>
</tr>
</tbody>
</table>
Continuous and Transient Control

- **Continuous Control**: L1s always attached to guest
- **Transient Control**: L1 attaches/detaches from guest as needed