Containers and Virtual Machines at Scale: A Comparative Study

Prateek Sharma
Lucas Chaufournier
Prashant Shenoy
Y.C. Tay
Virtualization in Data Centers and Clouds

- Data centers and clouds host multiple applications
- Virtualization and cluster management frameworks used for managing and multiplexing resources
- Virtualization options: Hardware-level and Operating System-level
# System Virtualization Today

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Machines</td>
<td>Containers</td>
</tr>
<tr>
<td>Example</td>
<td>Example</td>
</tr>
<tr>
<td>VMWare, Xen, …</td>
<td>Docker, LXC, …</td>
</tr>
<tr>
<td>Management</td>
<td>Management</td>
</tr>
<tr>
<td>OpenStack,…</td>
<td>Kubernetes,…</td>
</tr>
<tr>
<td>Performance</td>
<td>Performance</td>
</tr>
<tr>
<td>Noticeable overhead</td>
<td>“Lightweight virtualization”</td>
</tr>
</tbody>
</table>
Problem Statement

• Hardware virtualization predominant (especially in public clouds)
• OS virtualization (Containers) is being rapidly adopted

• How do they compare in multi-tenant data centers and cloud environments?
  1. What are the tradeoffs in application performance?
  2. How do they affect resource management and deployment?
  3. What is their influence on software development?
Outline

• Motivation and problem statement
• Virtualization background
• Performance comparison
• Managing and deploying applications
• Hybrid virtualization approaches
• Related work
• Conclusion
Hardware Virtualization

- *Hypervisor* exposes virtual hardware (CPU, memory, I/O)
- Applications run on top of a guest Operating System
- Multiple layers between application and hardware may reduce performance

![Diagram showing hardware virtualization with hypervisor, virtual hardware, guest OS, and applications.](image-url)
Operating System Virtualization

- Lightweight virtualization of OS resources and interfaces
- Applications run in Containers and share the underlying OS
- Cgroups in Linux for isolating resources (cpu, mem, I/O,…)
- Namespaces to isolate process hierarchies, sockets, filesystem,…

Diagram:
- Hardware
  - Operating System
    - Virtualization Layer
      - Application
        - Container
          - LXC, Jails, Zones
            - Linux, BSD, Solaris
Outline

• Motivation and problem statement
• Virtualization background
• **Performance comparison**
• Managing and deploying applications
• Hybrid virtualization approaches
• Related work
• Conclusion
Performance Evaluation Setup

- KVM for hardware virtualization vs. Linux Containers (LXC)
- Containers and VMs given same physical resources using cgroups
- CPU, memory, I/O intensive benchmarks
- In multi-tenant environments, applications can face interference
- Performance depends on type of co-located application:

  - Competing
  - Orthogonal
  - Adversarial (Intentional starvation)
CPU Isolation

- Container and VM CPUs pinned
- Containers can face starvation due to shared OS CPU scheduler
- **Shared OS or hypervisor components can reduce isolation**
- Map applications to servers carefully to minimize interference

Kernel compile slowdown relative to no-interference

- **VMs**
  - Competing Kernel-compile
  - Orthogonal File-bench
  - Adversarial Fork-bomb

- **Containers**
  - Did Not Finish

10-25% slowdown
I/O Isolation

- Equal allocation of I/O bandwidths using cgroups
- Disk I/O handled by hypervisor (virtIO in KVM)
- Both VMs and containers have shared I/O paths, causing interference
Outline

• Motivation and problem statement
• Virtualization background
• Performance comparison
• Resource management and deployment
• Hybrid virtualization approaches
• Related work
• Conclusion
Resource Overcommitment

- Cluster managers *overcommit* resources to increase efficiency
  - Allocate more resources than are actually available
- Hypervisors support overcommitment mechanisms for VMs
- OS overcommits container resources using soft-limits
  - Free (but allocated) resources can be used by other applications

- Memory overcommitted by 2x for SpecJBB
  - Excessive page faults in guest OS, because VM memory limits are hard
- **Performance depends on overcommitment capabilities of virtualization layer**
Migration

- Live-migration used for load balancing and consolidation
- VM migration implemented in all hypervisors
- Container migration is not production-ready
  - Involves careful preservation of OS state (process table, network sockets, open file descriptors, …)
  - Containers can instead be checkpointed and restarted

<table>
<thead>
<tr>
<th>Memory size (GB)</th>
<th>Container</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Compile</td>
<td>0.42</td>
<td>4</td>
</tr>
<tr>
<td>YCSB</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SpecJBB</td>
<td>1.7</td>
<td>4</td>
</tr>
<tr>
<td>File bench</td>
<td>2.2</td>
<td>4</td>
</tr>
</tbody>
</table>

- Container memory state is smaller
- VMs must also save guest OS and cache pages
Getting Applications Into Production

• Getting applications from development to production involves creating disk images

• Fast image creation enables rapid testing and continuous deployment

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>VM (Vagrant)</th>
<th>Docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>236</td>
<td>129</td>
</tr>
<tr>
<td>NodeJS</td>
<td>304</td>
<td>49</td>
</tr>
</tbody>
</table>

• Docker: 2-6x faster
Big Picture

- Performance depends on workload and neighbors
  - Interference because of shared resources
- Different application management and deployment capabilities
- Neither is best for *all* use-cases and scenarios
- **Can we get the best of both worlds?**
Outline

• Motivation and problem statement
• Virtualization background
• Performance comparison
• Managing and deploying applications
• **Hybrid virtualization approaches**
• Related work
• Conclusion
Containers Inside VMs

- Run containers inside VMs
- Isolation and security of VMs
- Image-management, provisioning of containers

Promising architecture which combines benefits of today's hardware and OS virtualization
Lightweight VMs

• Heavily optimized and stripped-down hardware VMs
• Make VMs more like containers
• **Isolation properties of VMs + low overhead of containers**
  • Intel ClearLinux, VMware Project Bonneville
• Directly access host file system. No virtual disk abstraction.

<table>
<thead>
<tr>
<th></th>
<th>Boot-time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container</td>
<td>0.3</td>
</tr>
<tr>
<td>ClearLinux</td>
<td><strong>0.8</strong></td>
</tr>
<tr>
<td>KVM</td>
<td>~10</td>
</tr>
</tbody>
</table>

• Fast boot-times enable VMs to be used in container pipelines
  • Quick deployment tests, rapid scaling,…
Related Work

• Felter et.al. — *Single* container vs. VM performance study
• Agarwal et.al. — Comparison of memory footprint
• Containers in HPC (Ruiz et.al), data processing (Xavier et. al)
• Alternate virtualization architectures — Unikernels, NoHype (Keller et.al)
Conclusion

• Containers and VM represent two choices in system virtualization
• Different tradeoffs in performance isolation, deployment
• “Right platform” depends on workload, environment, and use-case
• Nested containers and lightweight VMs are two promising hybrid strategies
• Fast changing landscape: change in performance and capabilities will require continuing evaluation
Thank You
Questions?

http://cs.umass.edu/~prateeks
Backup Slides
Baseline

Latency (ms)

Operations/second

YCSB Redis operations
CPU with different Cgroups setting

![Graph showing CPU allocation vs SpecJBB throughput]

- CPU-set
- CPU-share

**CPU allocation:**
- 25%
- 50%
- 75%

**SpecJBB throughput:**
- 0
- 50000
- 100000
- 150000
- 200000

---

25
More Interference
Today’s Virtualization

**Hardware Virtualization**
- Applications run inside *Virtual Machines*
- VMware, Xen, KVM,…
- Can choose own guest OS

**Operating System Virtualization**
- Applications run inside *Containers*
- Docker, Jails, LXC,…
- Applications share the OS
- Lightweight virtualization

- Cluster-wide management tools help users deploy applications:
  - Placement of applications
  - Horizontal scaling
Application Deployment

- VMs contain entire OS, and have larger images
- Docker stores only differences (application layer)

<table>
<thead>
<tr>
<th>Image size</th>
<th>VM</th>
<th>LXC</th>
<th>Docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>1.68 GB</td>
<td>0.4 GB</td>
<td>112 KB</td>
</tr>
<tr>
<td>NodeJS</td>
<td>2.05 GB</td>
<td>0.6 GB</td>
<td>72 KB</td>
</tr>
</tbody>
</table>

Docker: 2-6x smaller

- Copy & Deploy

Base

Install app

Ubuntu 14.04

App
Base line performance

Isolation: CPU, Memory, Disk, Network

Resource Overcommitment, migration

Image Creation

Sw Engg