CSE 291J Virtualization: Course Summary

Yiying Zhang
Outline

• Course summary

• Brief intro to major datacenters/clouds

• Hints for computer systems design

• Final project presentation this Thur
  • (6min presentation + 2min Q&A) * 10 groups

• Final project summary report due 3/20

• Course evaluation!
Operating System Architectures

- User-Mode
- Kernel-Mode

- Monolithic Kernel
- MicroKernel
- ExoKernel (Library OS)
X86 Difficulties and Possible Solutions

• Not all sensitive instructions are privileged with x86, i.e., non-virtualizable processor
  • These instructions do not trap and behave differently in kernel and user mode
• Hardware-managed TLB

• Emulate
  • Interpret each instruction, super slow (e.g., Virtual PC on Mac)

• Binary translation
  • Rewrite non-virtualizable instructions (e.g., VMware)

• Para-virtualization
  • Modify guest OS to avoid non-virtualizable instructions (e.g., Xen)

• Change hardware
  • Add new CPU mode, extend page table, and other hardware assistance (e.g., Intel VT-x, EPT, VT-d, AMD-V)
Protection Rings

- More privileged rings can access memory of less privileged ones
- Calling across rings can only happen with hardware enforcement
- Only Ring 0 can execute privileged instructions
- Rings 1, 2, and 3 trap when executing privileged instructions
- Usually, the OS executes in Ring 0 and applications execute in Ring 3

Image Source: [https://commons.wikimedia.org/wiki/File:CPU_ring_scheme.svg](https://commons.wikimedia.org/wiki/File:CPU_ring_scheme.svg)
Hardware Virtualization Support

Virtualization w/o hardware support

Virtualization w/ hardware support

Dune utilizing hardware support

Hypervisor

Guest OS

Guest Applications

Hardware w/o VT-x

Hardware w/ VT-x

Guest OS

Guest Applications

Non-root

Host Applications

Root

Dune Process

OS

Root Ring 0

Root Ring 3

Non-root

Guest Applications

Guest OS

Root Ring 0

Root Ring 3
VM <-> Containers

Virtual Machines
- VM
  - Guest OS
  - App
  - App
- VM
  - Guest OS
  - App
  - App

Lightweight VM (LightVM, Firecracker)
- Light VM
  - Small OS
  - App
  - App
- Light VM
  - Small OS
  - App
  - App

Unikernels
- Single-App VM
  - Small OS
  - LibOS
  - App
- Single-App VM
  - Small OS
  - LibOS
  - App

Secure Container (gVisor, Unikernels as processes)
- Container
  - LibOS
  - App
  - App
- Container
  - LibOS
  - App
  - App

Containers
- Container
  - Host OS
  - App
- Container
  - HW
Major Clouds and Datacenters
AWS

• Biggest market share, longest history
• Highest compute (and other service) options
  >= 136 instance types in 26 families
• Storage
  – Simple Storage Service (S3)
  – Elastic Block Service (EBS)
• Many other services
  – Lambda (serverless)
  – ECS/EKS (managed containers)
  – ...

Amazon

- Storage
  - Dynamo, S3, EBS
- Database/NoSQL
  - DynamoDB, Redshift, ElastiCache
- Network
  - Customized NICs, virtualization support
- Hardware
  - ASIC (Nitro), x86, ARM
- Resource management
  - Fargate, Kubernetes
- Execution environment
  - Disaggregated storage, many virtualization options
- Dataflow/analytics
  - EMR, Athena
- Application
  - Neptune (graph), SageMaker (ML), Kafka (streaming)
Azure

• Good integration with Microsoft products
  – Customers that are already using Microsoft products (e.g., having existing licenses)

• Many instance types and service types

• Moved from Windows to Linux
Microsoft

- Storage
  - Azure storage (erasure coding), Project Silica
- Database
  - SQL Server
- Network
  - RDMA, FaRM
- Hardware
  - x86, FPGA (Catapult)
- Resource management
  - Some research in using ML
- Execution environment
  - Disaggregated storage
- Dataflow
  - Dyrad, DyradLINQ
- Application
  - Project Adam (ML)
Google Cloud Platform (GCP)

- Latest among the three to come in play and smallest market share, but with good growth
- Cheapest among the three
- Fewest instance types, allows customized CPU/memory sizes
  - bill based on total CPU and memory usages, not on total instance time
- Native kubernetes support
- Good support for cross geo-regions
Google

- Storage
  - GFS, next-gen GFS, Intel Optane
- Database
  - BigTable, Spanner (Geo)
- Network
  - FatTree
- Hardware
  - Commodity + TPU
- Resource management
  - Borg, Kubernetes, Chubby
- Execution environment
  - Non-disaggregated, containerized
- Dataflow
  - MapReduce
- Application
  - TensorFlow (ML), Pregel (graph)

Reliability

Open Source
Facebook

- Storage
  - TAO, Haystack (photo)
- Database/NoSQL
  - Presto, MySQL, Cassandra, Memcache
- Network
  - Taiji (traffic), FBOSS, Wedge
- Hardware
  - Commodity, some ASIC
- Resource management
- Resource control
- Disaggregated (pods)
- Dataflow
- Application
- SVE (video), PyTorch (ML)

Move Fast and Break Things
Move Fast with Stable Infra
Common (and Important) Themes

- Scalability
  - Dist sys, local sys, networking
- Reliability
  - Failure, bugs, testing
- Security
  - Across customers, internal, regulations
- Manageability

Want to know more? Take CSE291H Modern Data Center Systems
Hints for Computer System Design

Butler Lampson
Systems Design

- The external interface (that is, the requirement) is less precisely defined, more complex, and more subject to change.

- The system has much more internal structure, and hence many internal interfaces.

- The measure of success is much less clear.
<table>
<thead>
<tr>
<th>Why?</th>
<th>Functionality</th>
<th>Speed</th>
<th>Fault-tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Does it work?</td>
<td>Is it fast enough?</td>
<td>Does it keep working?</td>
</tr>
<tr>
<td><strong>Where?</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Completeness</strong></td>
<td>Separate normal and worst case</td>
<td>Shed load</td>
<td>End-to-end</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End-to-end Safety first</td>
<td></td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>Do one thing well:</td>
<td>Make it fast</td>
<td>End-to-end</td>
</tr>
<tr>
<td></td>
<td>- Don’t generalize</td>
<td>Split resources</td>
<td>Log updates</td>
</tr>
<tr>
<td></td>
<td>- Get it right</td>
<td>Static analysis</td>
<td>Make actions atomic</td>
</tr>
<tr>
<td></td>
<td>- Don’t hide power</td>
<td>Dynamic translation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Use procedure arguments</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Leave it to the client</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Keep basic interfaces stable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Keep a place to stand</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>Plan to throw one away</td>
<td>Cache answers</td>
<td>Make actions atomic</td>
</tr>
<tr>
<td></td>
<td>Keep secrets</td>
<td>Use hints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use a good idea again</td>
<td>Use brute force</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide and conquer</td>
<td>Compute in background</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch processing</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Summary of the slogans
Keep It Simple

- **KISS: Keep It Simple, Stupid.** (Anonymous)

- *If in doubt, leave it out.* (Anonymous)

- *Exterminate features.* (C. Thacker)

- On the other hand,

- *Everything should be made as simple as possible, but no simpler.* (A. Einstein)
Making Implementation WORK!

- *Perfection must be reached by degrees; she requires the slow hand of time.* (Voltaire)

- *Plan to throw one away; you will anyhow*

- Use a good idea (and implementation) again instead of generalizing
Final Thoughts?