CMPSCI 677 Distributed and Operating Systems **Spring 2024**

Lecture 8: February 28

Lecturer: Prashant Shenoy *Scribe: Krishna Gudipaty (2024)*

8.1 Overview

In this lecture, we continued Cluster Scheduling i.e. Borg Scheduler and started a new topic, Virtualization.

8.2 Cluster Scheduling

8.2.5 Borg Scheduler

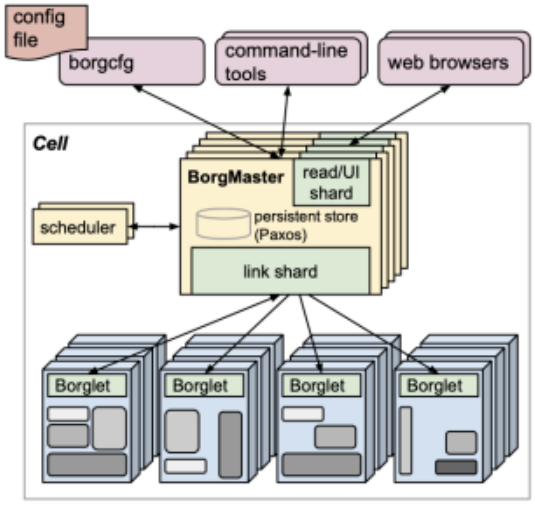


Figure 8.1: Borg Scheduler

This is Google's production cluster scheduler that was designed with the following design goals:

- The culture should scale to hundreds of thousands of machines.
- All the complexity of resource-management should be hidden from the user.
- The failures should be handled by the scheduler.
- The scheduler should operate with high reliability.

These ideas were later used to design Kubernetes. Borg is a mixed scheduler designed to run both interactive and batch jobs. Interactive jobs like web search, mail are user facing production jobs are given higher priority.

Batch jobs like data analysis are non-production jobs are given lower priority. Interactive jobs will not take up the entire cluster. The remaining cluster is used for batch jobs.

In Borg, a **cell** is a group of machines. The idea behind Borg scheduler is to match jobs to cells. The jobs are going to specify the resource needs and Borg will find the resources to run it on. If a job's needs change over time, it can ask for more resources. Unlike Mesos, the jobs do not wait for an offer but "pull" the resources whenever they need it.

Borg has built-in fault tolerance techniques. For e.g. if a cell goes down, Borg would move the job to some other cell. To allocate set of machines to a job, Borg would scan machines in a cell and start reserving resources on various machines. This is called **resource reservation**. The resources reserved are used to construct an allocation set. This allocation set is offered to the job.

Question: So instead of waiting, the framework can ask for more resources?
Answer: There is no notion frameworks in case of Borg. A job can ask for more resources. A job itself could be framework, batch job or a web application. So Borg is not necessarily doing a two-level allocation.

There is ability to preempt. Lower priority jobs are terminated to allocate resources to a higher priority job.

Question: After termination does Borg save the work in some memory. **Answer:** Typically Borg is not responsible for saving state, just resources. Developers might add checkpoint if their job is low priority

Question: Whats the point of a cell. **Answer:** Lets take a look at multi tiered applications. We need 3 machines for the Interface, Application, and Databse. Or a clustered webserver. Of course you can ask for just 1 machine.

Question: Are cells created proactivly or reactivly **Answer:** Generally "best fit" on set of machines. Don't have to take whole machine. Large services take a group of machines

Question: Can there be a single point of failure for interactive jobs. **Answer:** If the scheduler goes down the whole system goes down and if the application fails, there is 5* replication to avoid this. Generally seperated geographically at different data centers. The borg monitor applications and will restart cells or machines in the event of application failures. No state saves however.

Question: When you say terminate, are we talking about pausing a job or killing a job?
Answer: Termination is a policy decision. The important part of preemption is to reclaiming the servers. If the job being terminated is a web server, it doesn't make sense to pause it. But if is a long running batch job, it would make sense to pause it.

Question: Is the priority determined based on interactive and batch or can we have multiple priorities?
Answer: There is a clear distinction between interactive and batch. The idea can be extended to have multiple levels. Kubernetes allows you to make multiple priority levels.

The coordinator of Borg is replicated 5 times for fault tolerance. The copies are synchronized using Paxos(would be studied in greater detail in Distributed Consensus). It is designed with high degree of fault tolerance.

The scheduler has a priority queue. So the scheduler can either be best-fit or worst-fit depending on whether you want to spread the load or concentrate the load.

8.3 Virtualization

Virtualization uses or extends an existing interface to mimic the behaviour of another system. It is introduced in 1970s by IBM to run old software on newer mainframe hardware. It provided a software layer which

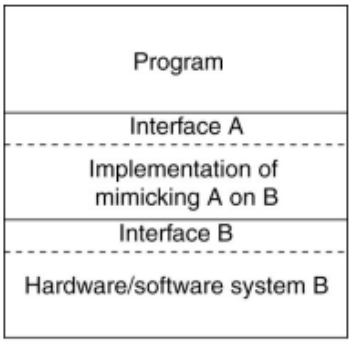


Figure 8.2: Virtualization

emulates an old hardware, so an old software could be run on top of the software layer.

Question: Do mainframe machines have an operating system?
Answer: Yes, OS/360 and many generations of it. Early versions of UNIX came out from those OSes.

Question: Is there a difference between virtualization and emulation
Answer: Emulation is a type of virtualization in some cases.

Question: What part of the image is the software layer?
Answer: The part between Interface A and B. That is the virtualization layer and it is implemented in software.

8.4 Type of Interfaces

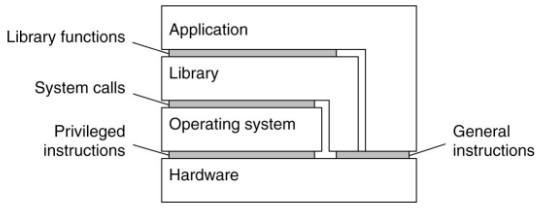


Figure 8.3: Types of interfaces in virtualization

Using virtualization, different layers of the hardware-software stack can be emulated.

- Assembly instructions (Hardware virtualization)
- System calls (OS-level virtualization): E.g., The open-source software Wine emulates Windows on Linux.
- APIs (Application-level virtualization): E.g., JVM

8.5 Virtualization

In native virtualization, one or more unmodified OSs and the applications they contain are run on top of virtual hardware, given that the underlying native hardware and virtual hardware are of the same type. Here the VM simulates "enough" hardware to allow the OSs to be run in isolation. One limitation is that one can run OSs designed for the same hardware only. One use case is that the virtual OS can be used as a sandbox for testing components in different environments.

8.5.1 Types of Virtualization

8.5.1.1 Emulation

In emulation, a software simulation of a particular type of hardware or architecture is done using another. Once you've the emulated hardware, you can run an OS on it. An example would be to emulate an Intel process on ARM hardware, and thereafter any OS that was compiled for ARM will run unmodified on the emulated Intel processor. The OS will execute machine instructions on the emulated hardware, which won't run as-is on the native hardware, and therefore binary translation will need to be performed to convert them to native instructions. This caused significant overhead/slowdown, which causes a performance penalty.

Examples: Box, QEMU (Used by much Linux software), VirtualPC for MAC (Before, MAC computers were PowerPC based, and VirtualPC emulated a x86 software layer on PowerPC machine, allowing one to run Windows)

8.5.1.2 Full/Native Virtualization

The VM does not emulate the entire machine, but it emulates enough hardware to run another system. This is typically done when an emulated interface and a native interface are the same. Since there is no translation of instructions it is much faster than emulation.

Applications of nature virtualization include: Virtualbox, Parallels **Examples:** IBM VM family, VMWare Workstation, Parallels, VirtualBox.

Question: What is difference between Emulation and Full Virtualization
Answer: Emulation is often different from the underlying interface. Think ARM to x86. In full virtualization the interfaces are the same.

- Running a completely different OS on top of a host OS.
- Acting as a sandbox for testing.
- Running multiple smaller virtual servers on a single server.

8.5.1.3 Para-virtualization

Similar to full/native virtualization except that the kernel of the guest OS is modified so that it uses special APIs to call the hypervisor instead of directly accessing the hardware. The applications run are unmodified.
Examples: Xen, VMWare ESX Server.

8.5.1.4 OS-level Virtualization

Here the OS allows multiple secure VMs to be run on top of a native OS kernel. Each application run on a VM instances sees an isolated OS. This is lightweight as different OSs are not run.

Examples: Solaris Containers, BSD Jails, Linux Vserver, Linux containers, Docker.

Question: What is the difference between OS and Hardware level virtualization.
Answer: We have some interface A used to mimic interface B. Hardware is emulating the actual chip.

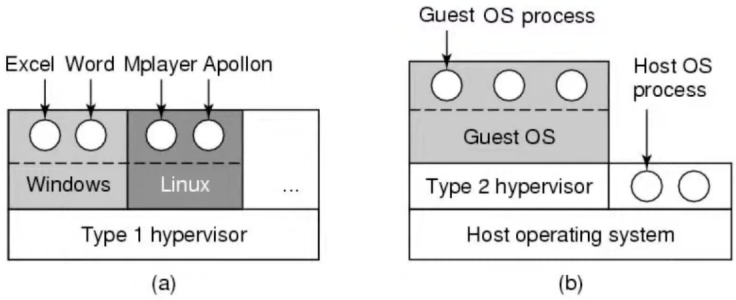
8.5.1.5 Application level virtualization

This type of virtualization uses an application interface to emulate another application. Here, the application is given its own copy of components that are not shared like global objects, registry files etc.

Examples: JVM written using C libraries exports the JAVA interface. Rosetta on Mac (also emulation) allows us to run binaries compiled on one architecture on another architecture WINE.

8.6 Hypervisors

The hardware virtualization layer is called a hypervisor or a virtual machine monitor (VMM). A hypervisor is the virtualization layer, which takes care of resource management, isolation and scheduling. There are 2 types of hypervisors: type 1 and type 2



8.6.1 Type 1 Hypervisor

Type 1 hypervisor runs on "bare metal". It can be thought of as a special OS that can run only VMs as applications. On boot, it's the hypervisor that boots.

Question: Do the multiple OSs running expect the same hardware?
Answer: Yes, because the hypervisor gives the illusion that the OS-s is running on the underlying native hardware. The hypervisor allocates resources between VMs, schedules VMs, etc., the hypervisor behaves like a pseudo-OS for the VMs.

A Type 2 hypervisor runs on a host OS and the guest OS runs inside the hypervisor.

Question: Does type 1 belong to native or paravirtualization?
Answer: Both.

8.6.2 Type 2 Hypervisor

The boot process boots the native OS, the hypervisor runs as a application on the native OS. The native OS is called host OS and the OSs that run on the hypervisor are called guest OSs.

Question: Can you do further virtualization?
Answer: Yes, but it is complicated. It is called nested virtualization

Question: Can multiple guest OSs be run?
Answer: Yes, multiple VMs can be run, each with its own guest OS

Question: Is the bootloader GRUB a hypervisor?
Answer: No, GRUB used in dual boot machines decides which one OS it has to boot, whereas hypervisors can run multiple guest VMs.

Question: Can a single hypervisor simulate multiple hardwares?
Answer: No, for that emulation needs to be used.

Question: In an emulator, if a VM has the same processor as host machine, will it be faster?
Answer: Yes, because the emulator will not need to emulate another processor and will default to full virtualization instead. Thus reducing the overhead and making it faster.

Question: Why would we want to start a VM with the same OS as the underlying host OS?
Answer: The VM can be used as a second machine for multiple reasons. For eg.: for testing purposes. However, the guest OS need not necessarily be the same.

Question: If the guest OS gets affected by malware, will it also affect host OS?
Answer: The host and guest OSs are separate OS environments on their own. Thus, if guest OS is bridged, it will not have an impact on the host OS.

Question: Why would we want to use Type 1 hypervisor and not install an OS?
Answer: If the requirement is to run multiple VMs on a single machine, a hypervisor should be installed. It will allow a large server to run multiple smaller servers each capable of running different applications. If a single OS is to be used on the machine, then virtualization is not needed and hypervisor should not be installed.

Question: Do Data centers use Type 1 hypervisors?
Answer: Yes, Type 1 hypervisors are very commonly used in data centers and cloud platforms.

Question: Can one VM span multiple physical machines?
Answer: No, because an OS cannot run on different machines. However, multiple VMs running on different servers can communicate with each other to act like a distributed system.

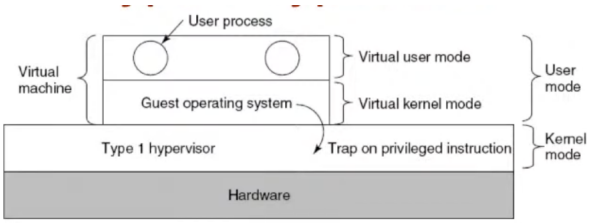
Question: Do guest OSs communicate with each other via hypervisor?
Answer: Guest OSs have their own network cards with individual IP addresses and can thus directly communicate with each other over network. Since each VM is a software emulated version of a full PC, mechanisms such as shared memory cannot be used for communication between them.

8.7 How Virtualization Works?

For architectures like Intel, different rings signify different levels of privilege. The CPU can run in either user mode (ring 3) or kernel mode (ring 0). In kernel mode, any instructions can be run. But in user mode, only a subset of instructions is allowed to run. The OS kernel is trusted more than user applications, so the OS

kernel is run in kernel mode and user applications are run in user mode to limit instruction sets. A simple example is the halt instruction which user applications are not allowed to run. Also, memory management instructions are only allowed to run in kernel mode. Intel CPUs have intermediate rings (ring 1, ring 2) in which guest OSes can be run. Sensitive instructions (I/O, MMU, halt, etc.) are only allowed to run in kernel mode. Privileged instructions cause a trap or interrupt. I/O is one example. These are not related to user mode or kernel mode.

8.7.1 Type 1 Hypervisor

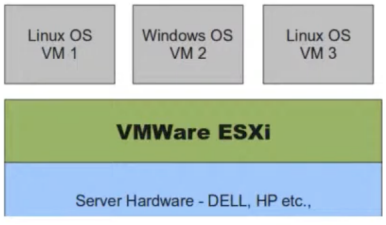


Theory: Type 1 virtualization is feasible if sensitive instruction is subset of privileged instructions or all sensitive instructions always cause a trap.
Reasoning: On booting a Type 1 hypervisor, it runs in kernel mode. A Windows VM run on the hypervisor should not be trusted as much as the hypervisor and is therefore run in user Mode. Windows assumes it is the kernel and can run sensitive instructions, but these sensitive instructions won't run because it will be running in user mode. The solution is that the hypervisor intervenes and runs each sensitive instruction attempted by the Windows VM. How will the hypervisor be alerted when Windows attempts so? If the sensitive instruction causes a trap, the hypervisor intervenes and executes it for the VM.

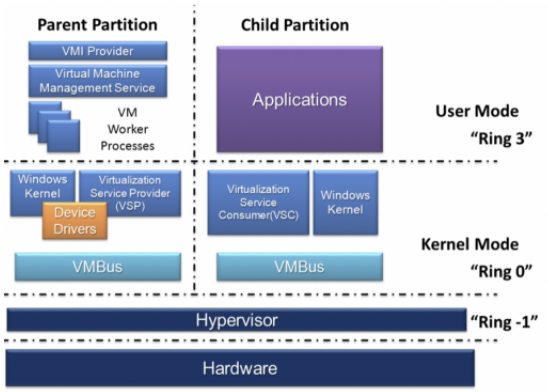
Early Intel processors did not have Type 1 support as they simply ignored any sensitive instructions executed in user mode. Recent Intel/AMD CPUs have hardware support, named Intel VT and AMD SVM. The idea is to create containers where a VM and guest can run and that hypervisor uses hardware bitmap to specify which instruction should trap, so that sensitive instruction in guest traps to hypervisor. This bitmap property can also be turned off.

Question: Are there instructions that are privileged but not sensitive?
Answer: Yes. Eg: Divide by zero or an illegal pointer access. All these will generate an exception.

Examples:



a) VMWare ESXI running, a specialized OS kernel that can run any arbitary VMs on it



https://en.wikipedia.org/wiki/Hyper-V

b) Windows Hyper-V creates partitions, runs Windows in the parent partition (one copy of windows is mandatory), and the child partitions can run Linux or any other OS. Less flexible than ESXI.

c) Linux KVM or kernel virtual machine: Implemented as a device driver that gives barebone support for Type 1. Along with some other components (like QEMU) gives the functionality for Type 1 hypervisor. Linux OS has KVM loaded as a module. Thus, Linux OS is a hybrid OS which can be booted normally on a machine and can serve as a standard OS and also as a hypervisor simultaneously when KVM is enabled.

Question: In the above Hyper-V diagram, does the "Ring -1" indicate that the hypervisor runs with negative privileges?
Answer: No, there is no concept of negative privileges. It just indicates relative privileges.

Question: Can a physical machine run more than one ESXi hosts?
Answer: No, a physical machine can boot only one OS at a time, either a standard OS or the hypervisor unless there is hardware partitioning configured. But normal machines don't have that and can only run a single OS.

Question: Can Linux KVM be classified as Type 2 Hypervisor?
Answer: Since the KVM hypervisor is a part of the OS and not run as a separate application on top of the OS, it is classified as Type 1.

Question: How is a VM started in KVM? Do we boot host OS first or the VM can be started directly?
Answer: To start a VM, we always boot the hypervisor first. In Linux KVM, since the hypervisor is a part of the OS, we boot Linux OS first and enable KVM. VMs can be started on it now.

8.7.2 Type 2 Hypervisor

A Type 2 hypervisor runs as an application on the host OS and therefore does not have kernel level privileges. Again, only the host OS can run in kernel mode. The Guest OS can also run only user mode instructions.

Therefore, the Type 2 hypervisor performs *dynamic code translation*. It scans instructions executed by the guest OS, replacing the sensitive instructions with function calls. These function calls invoke the Type 2 hypervisor which in turn makes system calls to the OS, thereby involving the host OS. This process is called binary translation. This leads to slowdown as every piece of sensitive code has to be translated. Therefore, VT support is not needed, no support is needed from the hardware to ensure that all sensitive instructions are privileged.

For example, VMWare Fusion, upon loading program, scans code for basic blocks and replaces sensitive instructions by VMWare procedure using binary translation. Only the guess OS's instructions need to be scanned, not the applications.

Question: How do applications that do not support VMs identify that they are running on a VM?
Answer: A VM simply looks like any other physical machine to an application. Thus, there is no technical reason as to why an application cannot run on a VM. However, some applications are not licensed to run inside VMs. Eg: MacOS. Applications can parse some markers to identify parameters such as drivers used by Type 2 hypervisors, but this is not a robust way.

Question: What happens if guest OS has hardware requirements that the physical machine does not satisfy?
Answer: This impact will be similar to when such a situation occurs in a non-virtualized environment. Eg: An OS might not boot if it has less resources.

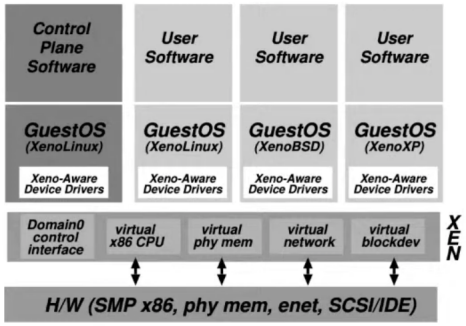
8.7.3 Para-virtualization

Instead of dynamically translating the sensitive instructions, leading to overhead, we can categorically change the kernel instructions to replace the sensitive instructions with hypercalls (call to the hypervisor) to get a modified OS with no sensitive instructions. Every time a sensitive function needs to be executed, a hypercall is made instead.

Both Type 1 and 2 hypervisors work on unmodified OS. In contrast, para-virtualization modifies OS kernel to replace all sensitive instructions with hypercalls. Thus, the OS behaves like a user program making system calls and the hypervisor executes the privileged operation invoked by hypercall. Since the modified OS eliminates all sensitive instructions with hypercalls, it can be run on any processor as hypervisor will handle invoking system calls. Paravirtualization is the only way to support Type 1 virtualization on older processors(which do not generate traps for sensitive instructions).
Note that such a modified OS will always need hypervisors and will no longer support standard non-virtualized environments.

Question: In a Type 2 hypervisor, after replacing sensitive instruction with a procedure, will it make a trap to the host OS?
Answer: It will not generate a trap, because the hypervisor will invoke a system call to directly ask the host OS to execute an instruction.

Question: What is the difference between a system call and a kernel trap?
Answer: A system call is essentially a function call to the OS through which which you can ask the processor to perform an operation whereas a trap is like an interrupt to the processor which the processor will have to serve by identifying what caused it.



Example: Xen ran as a para-virtualized version of Linux when the hardware did not support Type 1 hypervisors. Xen needed Linux to run in domain 0 (master partition or controller VM) just like HyperV. All OS drivers are placed in this VM(also called driver domain). So any other VM wanting to make a driver call will direct it to the driver domain via the hypervisor.

Question: Does this need communication between VMs?
Answer: Yes, but this communication is not network-based and can happen via hypervisor through IPC(Inter-Process Communication).

Question: Will para-virtualization have faster execution at the cost of having a modified OS?
Answer: It will definitely be faster than Type 2 virtualization.

8.8 Virtualizing Other Resources

8.8.1 Memory Virtualization

The hypervisor has control over RAM and allocates memory to the guest OS, and the guest OS allocates memory to its processes. If the guest OS tries to change the memory allocation of a process, that is a sensitive instruction because only the host OS is allowed to change page tables. Processes are not allowed to this. But a guest OS in Type 1 does not have those privileges. It still maintains page tables, and it still thinks it owns the machine that it can do whatever it wants, but it is not allowed to do so.

So what we do here is that we change those page tables in guest OSes to be read-only. When OS tries to write to that page table, a trap is created because that is a write instruction to read-only memory page. The hypervisor maintains a second page table which is called a shadow page table. That is a mirror of the original page table. It makes those changes in its actual page table. It utilizes the existing hardware feature that causes a trap when a write instruction happens on read-only region.

8.8.2 I/O Virtualization

Typically, the OS manages the physical disk, the guest OS should not be able to make changes to it. The hypervisor creates a large file on the guest's file system that emulates a disk called virtual disk (eg., vmdk for VMWare) When the guest OS writes to the disk, it effectively writes to this virtual disk. Multiple virtual disk maps to the real disk.

Question: Does a virtual disk have 1:1 mapping or 1:N mapping to VMs?
Answer: Each VM will have its own virtual disk. Multiple VMs cannot be sharing the same virtual disk as theu might overwrite contents causing discrepancies.

Question: In a shadow page table, do we map guest physical address to host physical address?
Answer: Each VM has its own address space. Page tables inside the guest OS will keep mappings of this address space. However, this address space will translate to some memory addresses in the host space as well. The shadow page table holds these mappings.

Question: Does each VM have one shadow table?
Answer: Page tables are per-process. So each VM will have multiple shadow tables based on the processes running in them.

Question: Can Type 2 hypervisors leverage VT technology?
Answer: There can be a hybrid implementation of Type 2 hypervisor to leverage VT technology if the

underlying processor supports it. This reduces the translation overhead for sensitive instructions and thus make it faster.

8.8.3 Network virtualization

Similar to I/O virtualization, a network card is also virtualized. Here, VMs get a software emulation of the physical ethernet card. Whenever a read/write happens on this card, it creates a trap and the operation is redirected to the physcial ethernet card instead. The logical(emulated) network card cannot communicate on its own. It can only receive requests and route them to the physical card to process them.

8.9 Benefits of Virtualization

One major benefit is that virtualization makes it easier to distribute pre-built applications and software. One can design virtual appliances (pre-built VM with pre-installed OS and pre-configured applications) that are plug-and-play.

For multi-core CPUs, careful allocation of CPU resources per VM can be made.

8.10 Use of Virtualization

a) Cloud Computing b) Data Centres c) Software Testing and Development

Examples: IBM VM family, VMWare Workstation, Parallels, VirtualBox.