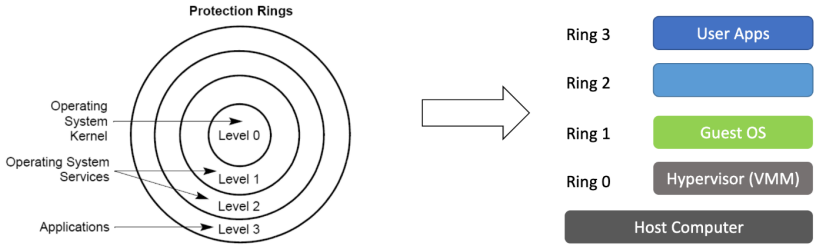
Cloud Computing Midterm 1

Disha Shetty

Q1)

1) Please describe the high-level design ideas to virtualize CPU, given that all of the sensitive instructions are the privilege instructions?

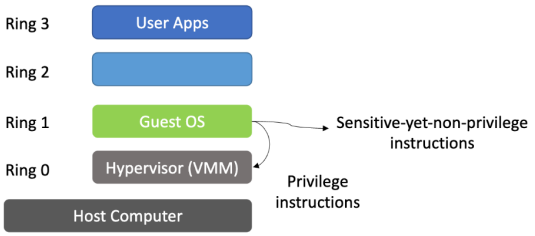
Answer:



Several virtual machines that may each run their own operating system and applications are created in order to virtualize a CPU. Certain instructions can only be executed in the Ring 0 level, hence they cannot be directly executed by Guest OS. If the Guest OS is kept in Ring 0, it may not coordinate with native/host OS. Problems will arise with respect to security breaches, processes will be killed etc. Hence, we cannot give full privileges to the Operating System. One solution we can come up with is to keep the hypervisor in Ring 0 and place the guest OS on top of it (Refer the top right side of the diagram). This will cause the trap/execution in privilege level other than the Ring 0. Later the host i.e. the hypervisor/VMM can interpret the instructions and once the OS gets notified it will then modify the privileged instructions.

2) Please describe the high-level design ideas to virtualize CPU, given that only part of the sensitive instructions are the privilege instructions?

Answer:



The above scenario is only with respect to if sensitive instructions are privileged instructions. But this is not a perfect world. There might be a scenario wherein they may not be the same. Sensitive instructions, unlike privileged instructions, won't affect other processes, but they

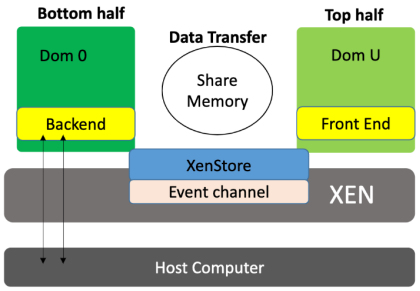
shouldn't be allowed to directly execute in VM virtualization because they might be able to interfere with the operations of the VM. The level remains the same: Ring 0 will be the hypervisor and Ring 1 will be the Guest OS. One solution is that for sensitive but non-privileged instructions we can do an emulation - binary translation as known as trap-and-emulate; before sending the instructions to the CPU, we can capture it and then the interpreter will interpret the instructions and convert them into binary code. In this case, the interpreter will act like a hypervisor and it traps the sensitive instructions.

Q2)

1) Please state the main sources that negatively impact I/O performance in KVM or Xen based virtualization solutions in comparison with the native (i.e., no virtualization).

Answer:

Xen Split driver model:



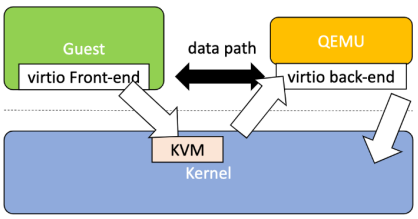
The original drivers are split into 2 parts: top half is the guest VM and the bottom half is Dom 0 in which a shared memory mechanism is provided for data communication. It uses a round robin scheduling approach, which has a large impact on the I/O latency. As the packets arrive, there is a high chance that delay will be proportional to the number of runnable domains. It is then transmitted in the queue, only if DOM 1 is operational and hence it will be delayed depending on the VM. We can go ahead with the optimizations of i/o intensive VMs. One can be a **small time scheduling time slice**. If we do have the small time slice, the context switch frequency increases which results in high overhead.Second is **running I/O domains with high priorities**.This is due to the possibility that I/O-intensive workloads could frequently interrupt CPU-intensive activities, causing a high number of context switches and a high level of overload.This could result in new issues like a cold start penalty for the following process.

2) To achieve high I/O performance under Xen/KVM virtualization, what are the possible optimization solutions, and why do you think these solutions will work? (name three)

Answer:

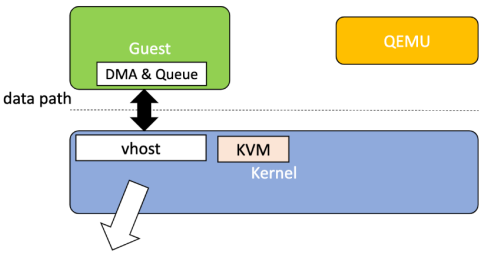
Below are the possible solutions to achieve high I/O performance under Xen/KVM virtualization:

1) Para-virtualization:



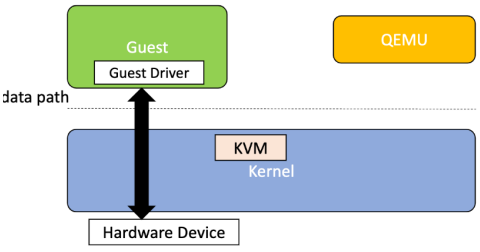
As given in the diagram, here the front end is guest VM and backend is QEMU.Here the request goes through QEMU and in a simple manner the QEMU emulates back I/O access which reduces the overhead unlike the previous method.

2) vHost:



Normally, the QEMU user space process emulates I/O accesses from the guest.
In this case, vHost removes QEMU and adds a virtual environment into the kernel. This enables device emulation code to make direct calls to kernel subsystems rather than using user space to make system calls.

3) Hardware Support



This feature allows an administrator to assign I/O devices to VMs in any desired configuration. It allows bypassing the kernel to directly access VT technology. But this

cannot function without the Hardware support. It supports address translations for device DMA data transfers and provides VM routing and isolation of device interrupts.
3) Under a round-robin vCPU scheduler, suppose the time-slice (i.e., maximum time that a vCPU can run) is 30 ms. Four VMs (VM1 to VM4, each with 1 vCPU) share a single CPU. When a client sends an I/O request to VM1, how long does the client expect to receive the response from VM1 in the worst case?

Answer:

Since each VM have a time slice of 30 ms,
IO wait time = No. of virtual machine * schedule time slice
IO wait time = 4 * 30 = **120 ms**
Thus, in the worst case, the client can expect to receive response in 120ms

What are the results if the time-slice is 10 ms and 1 ms, separately?

IO wait time = No. of virtual machine * schedule time splice
= 4*10 = **40 ms**

IO wait time = No. of virtual machine * schedule time splice
= 4 * 1 = **4 ms**

What are the gains and losses using a small time-slice (e.g, 1 ms) in comparison with a large one?
In the case of a large time slice, there is a possibility of I/O delay and in a small time slice, the context switch frequency increases which results in high overhead. This then arises to the cold start penalty for the upcoming process.

Q3)

1) How does the OS kernel support containerization?

Answer: The OS kernel provides the necessary features and abstractions to support containerization, which is a lightweight form of virtualization that allows multiple isolated user-space instances (containers) to share a single host operating system.

Here are some ways in which the OS kernel supports containerization:
Namespace isolation: The kernel supports different types of namespaces that can be used to isolate various system resources, such as network interfaces, process IDs, user IDs, and file systems, among others. Each container can have its own set of namespaces, which provides process-level isolation between containers.
Control groups (cgroups): Cgroups are a kernel feature that allows administrators to limit and prioritise system resources such as CPU, memory, and I/O for a group of processes. Each container can be assigned to its own cgroup, which ensures that it gets a fair share of resources and prevents it from consuming too much.
Filesystem isolation: Containers often have their own file system namespace, which means that they can have a private view of the file system that is separate from the host and

other containers. This allows containers to have their own set of files and directories, and ensures that they cannot access or modify files outside of their namespace.

2) Can you guess what is the key functionality in Docker?
   Answer: Containerization is the key functionality in Docker.

3) Why is I/O performance under containerization much better than KVM or Xen based virtualization?
   Answer: Performance of the container is very similar to that of the native OS, running applications inside the OS is same as that of running processes and they are regulated in control groups. Since XEN and KVM are system-level virtualization methods, they must also have system-level components like the CPU scheduler, MMU, etc. I/O latency results from the above mentioned responses.

4) Then would the I/O performance of containers be the same as the native case (i.e., running processes directly on an OS), and why?
   Answer: Yes it will be the same because running a container is similar to running an OS. Hence, the performance of the container will closely be the same as that of the native case.

5) Containers are much light-weight in comparison with machine-level virtualization solutions like VMs. However, in practice (e.g., in public clouds) we do not completely replace VMs with containers. Why is that?
   Answer: Containers, according to me, are not at all safe as they share the same host kernel which in turn leads to weak isolation (cloud has to be isolated), which can break the current isolation and has easy access to the OS.

   Q4)

   1) gVisor (Google's container solution) provides another isolation mechanism for container, which intercepts application system calls and acts as the guest kernel. Instead of passing system calls to the native kernel, gVisor implements a substantial portion of the Linux system surface. Thus each container may have its own user-level kernel. This design supposes to have better security properties than traditional ones (e.g., Docker containers). How does gVisor achieve this?

   Answer:
   The major part of gvisor is the sentry wherein all the system calls are redirected to Sentry. Sentry usually makes host system calls to support all the operations, but does not allow the application to directly control the system calls. Ptrace is usually used in this which is a way to intercept the calls.

   2) Actually, gVisor trades performance for such better security properties. Which types of applications do you think suffer less in terms of performance drop when we use gVisor containers, and why? Do you have some ideas to mitigate performance overhead of gVisor?

   Answer:

It is true that gVisor adds an additional layer of isolation and security to containerized applications, which can result in some performance overhead. However, the exact amount of overhead will depend on the specific application and workload.

That being said, certain types of applications may be better suited for running in gVisor containers, and may suffer less in terms of performance drop. These include:

Stateless applications: Stateless applications, such as web servers, are designed to handle requests independently of each other. This means that they are often highly parallelizable and can be easily scaled horizontally. Since gVisor containers can run multiple instances of an application on a single host, this makes it easy to horizontally scale these types of applications without significantly impacting performance.

Microservices: Microservices are a software architecture pattern that involves breaking down large, monolithic applications into smaller, independently deployable services. Since gVisor containers provide additional isolation and security, this can be especially beneficial for microservices that handle sensitive data.

The gVisor mitigates but does not eliminate the security vulnerability. e.g., it is possible that if any one of the gVisor containers breaks out, it can allow unauthorised access across containers running on the same host.

Q6)

1) What are the key motivation that you think for many IT companies (e.g., Netflix, Amazon, Uber) to move their services to a microservices based architecture?

Answer:
Scalability: Microservices make it possible to scale applications or services efficiently at the component level rather than having to scale the complete monolithic program. Cost reductions and increased performance may result from this.

Flexibility: Because each service can be developed and delivered independently, microservices architecture enables for flexibility in both development and deployment. As a result, the application can be updated and iterated upon more quickly.

Resilience: By using microservices, if one component fails, the entire application may not be negatively impacted. This could enhance the system's overall availability and resilience.

2) In which situation(s), you prefer NOT to use microservices based architecture for designing your applications?

Answer:
Small Applications: Microservices are designed for complex and large-scale applications where modularity is essential.

Tight Budget: If you have a tight budget, microservices may not be the most cost-effective solution. It consumes additional resources, time consuming for design and deployment as well as external maintenance charges.
Technical Expertise: Microservices require a deep understanding of distributed systems and complex architectures. If you have limited expertise, it is better to opt for simple architecture.

3) What is the key goal that Kubernetes wants to achieve?

Answer:
Kubernetes provides the platform for managing containers which in the future reduces the complexity and also minimises the management of large scale container deployments. It also helps to ensure that applications are deployed and managed in a consistent and reliable way, which can help to improve the overall reliability and availability of the applications.

4) Can Kubernetes choose VMs as the hosting service (instead of containers)?

Answer: Yes, Kubernetes can choose VMs as the hosting service instead of the containers as they are primarily designed to manage and deploy containerized applications.

5) Kubernetes uses “Pod” as the basic abstraction. Why doesn’t it use “container” as the basic abstraction?

Answer: Kubernetes uses the "Pod" as the basic abstraction rather than the "container" because Pods are a higher-level abstraction and provide additional features and flexibility.

6) What is the “service” object used for in Kubernetes?

Answer: Service objects in Kubernetes are the YAML file.

7) How does Kubernetes provide high availability?

Answer: Kubernetes provides high availability through several mechanisms that ensure that applications deployed on a Kubernetes cluster remain available even in the event of node failures or other issues.