VMs vs Containers: a performance comparison (literature review)

Introduction

Cloud computing has fundamentally transformed the landscape of information technology by providing scalable, on-demand access to computing resources over the internet. This has enabled organizations to reduce capital expenditures, enhance operational efficiency, and accelerate innovation [3].
At the heart of cloud computing are virtualization technologies that abstract and pool physical resources, allowing multiple workloads to run concurrently on shared hardware. Virtualization is a technology that enables the creation and use of virtual versions of physical resources such as computers, servers, storage devices, or networks. Instead of operating directly on physical hardware, virtualized systems function on a layer of abstraction created by software [1].
Two predominant forms of virtualization have emerged in this space: virtual machines (VMs) and containers [1, 3].
Virtual machines have long been the cornerstone of cloud infrastructure but containers have risen to prominence more recently as a lightweight alternative to VMs.
Understanding the performance differences between VMs and containers is crucial for optimizing cloud deployments. This literature review will compare these technologies in terms of performance, resource utilization, scalability, and efficiency, providing insights to help organizations make informed decisions in their cloud computing strategies.

What are virtual machines?

A Virtual Machine (VM) is a software emulation of a physical computer that allows multiple operating systems to run simultaneously on a single physical hardware host. VMs operate using a hypervisor, which is a layer of software that sits between the hardware and virtualized operating systems, managing resource allocation and ensuring isolation. There are two types of hypervisors: Type 1 (bare-metal), which runs directly on the host hardware, and Type 2 (hosted), which runs on top of an existing operating system. Hypervisors like VMware ESXi, Microsoft Hyper-V, and KVM are examples of Type 1, whereas Oracle VirtualBox is an example of Type 2[1].

What are containers?

Containers are lightweight, portable units of software that package an application along with its dependencies, libraries, and configuration files into a single bundle that can run consistently across different computing environments. Unlike VMs, which emulate entire operating systems, containers use operating system-level virtualization and share the host system's kernel. This allows multiple containers to run on the same host without the need to replicate the overhead of full operating system instances, which significantly reduces the required computational resources [1, 3].

Containers are built upon a foundation of isolation technologies, such as Linux cgroups (control groups) and namespaces, which ensure that each container has its own isolated user space, resource limits, and network interfaces [3]. By sharing the underlying host kernel, containers achieve higher efficiency, faster startup times, and lower resource overhead compared to virtual machines. Popular containerization platforms include Docker and Podman, while Kubernetes is a widely-used orchestration tool that helps manage, deploy, and scale containers.

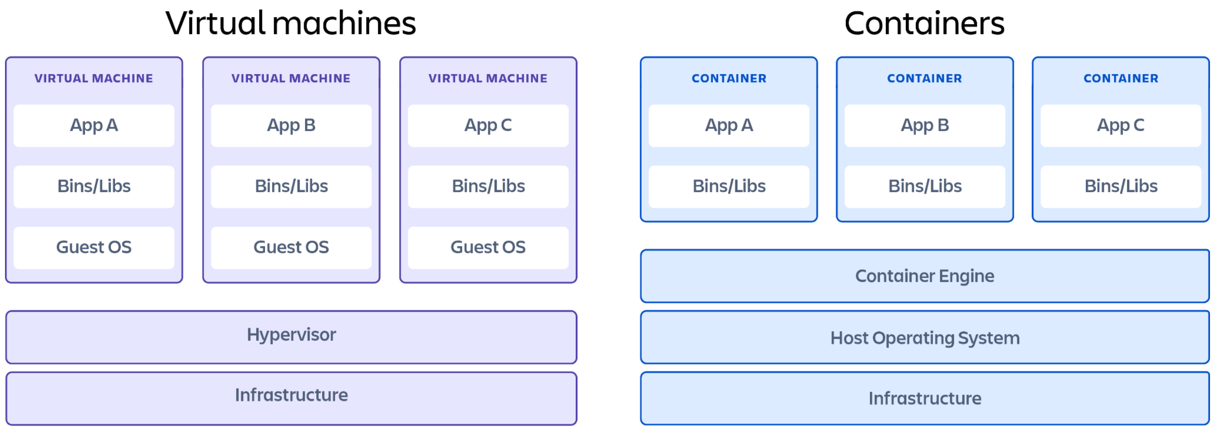


Figure 1: Architecture of virtual machines and containers [1]

Methodology

To conduct this literature review, we need to read peer-reviewed research papers and articles written by experts within the field of computer science, and specifically virtualization and its different branches. Therefore we choose five sources that comply with our above mentioned requirements, and that aim to do a performance comparison of Virtual Machines (VMs) and Containers. Here is a list of the sources:

- *Containers vs. virtual machines*, by Ian Buchanan [1].
- *Containers and Virtual Machines at Scale: A Comparative Study*, by P. Sharma, L. Chaufournier, P. Shenoy and Y.C. Tay [2].
- *A Comparative Study of Containers and Virtual Machines in Big Data Environment*, by Q. Zhang, L. Liu, C. Pu, Q. Duo, L. Wu, W. Zhou [3].
- *Containers & Virtual machines: A performance, resource & power consumption comparison*, by Martin Lindström [4].
- *Comparative Study of Virtual Machines and Containers for DevOps Developers*, by S. Deochake, S. Maheshwari, R. De, A. Grover [5].

The results that we present below are based on the different results that were achieved by the different sources above supported by graphs and figures to solidify clarity and illustrate how these technologies compare when it comes to different scenarios related to their performance.

Results

Single Machine and Single VM / Containers

a. Initial Human Efforts and Bootup Efficiency

Several parameters were evaluated that impact infrastructure expenses, including the time necessary to set up the infrastructure and the time required to start and stop the virtualization services, as an effort to determine to accurately compare the amount of human effort required to set up a VM and a container. Several tests were run to assess how long it took to set up and stop virtual machines and containers [5]. A simple Docker image for a container can be created and pushed in an average of 40 ms for the start time and 28 ms for the stop time thanks to the layered file system that Docker containers use for image generation. In contrast, the test showed that virtual machines typically take 55 seconds to start and 30 seconds to stop, the reason being the requirement of an installation of the complete operating system from scratch each time a new virtual machine is created. The test results unequivocally demonstrate that Docker containers start and terminate over 1000 times quicker than virtual machines. Invoking a system call to start a Docker container only takes milliseconds since Docker containers operate as processes inside the host operating system.

However, every virtual machine has to go through the complete generic operating system boot process, which takes tens of seconds. This led us to the conclusion that Docker containers are the superior option for reducing initial human effort and bootup efficiency since they significantly reduce the amount of time needed to start or stop the systems.

b. Disk I/O Operations

Randread and randwrite operations on the disk within virtual machines and Docker containers were conducted with a workload generator and the disk I/O performance of virtual machines, containers and bare metal servers as a reference point were compared. [5]. Containers, as shown in the figure, have substantially reduced overhead and offer performance that is equivalent to that of a bare metal system. On the other hand, because of their high resource cost and overhead, virtual machines perform poorly compared to Docker containers and bare metal servers. This led us to the conclusion that Docker containers are the superior option for I/O-intensive applications.

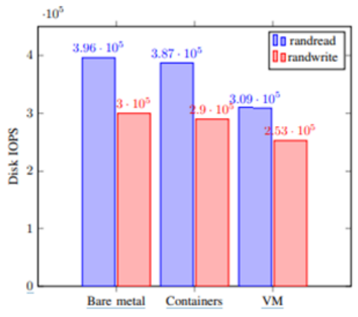
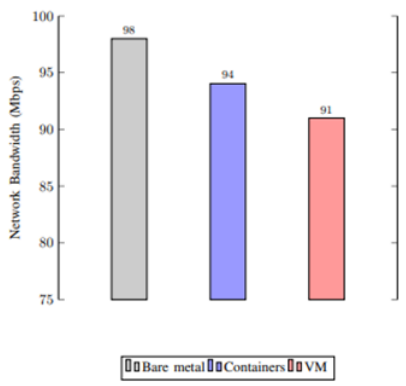


Figure 2: Disk I/O Performance Comparison

c. Network Bandwidth

netperf, a network benchmark program was utilized to monitor the network bandwidth and evaluate network latency. The outcomes of running netperf on bare-metal servers, virtual machines and containers were contrasted [5]. Figure 5 makes it clear that containers provide performance comparable to bare-metal, but virtual machines perform badly in terms of networking bandwidth falling behind containers by 3.2% and bare-metal servers by 7.1%. The average delay measurement for virtual machines and containers is shown in Figure 6. From the figure, we can deduce that containers are comparable to bare-metal but significantly better than VMs, the difference being 38.7% in TCP protocol and 29.1% in UCP protocol. This led us to the conclusion that Docker containers are a better option than virtual machines for web applications.



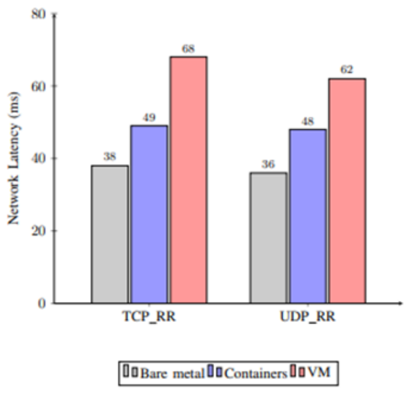


Figure 3: Network Bandwidth Comparison   Figure 4: Network Latency Comparison

d. CPU

The performance of virtual machines and containers were put into a test in terms of Floating-Point Operations per Second (FLOPS) utilizing Intel's Linpack tool to assess CPU performance, which gauges CPU through conducting heavy floating point operations [5]. The results indicate that containers leverage native memory swapping for heavy floating-point computations and the host operating system's native system calls, therefore they perform better than virtual machines. On the other hand, since for VMs hypervisor translating the system call to the hardware, virtual machines tend to perform poorly. The figure shows that containers run better than virtual machines using CPU benchmark performance on the Intel Core i7 Skylake CPU. Compared to bare metal machines as a benchmark, containers offer adequate CPU isolation with negligible to no overhead compared to VMs. This led us to the conclusion that Docker containers are a better option than virtual machines for CPU heavy operations or applications.

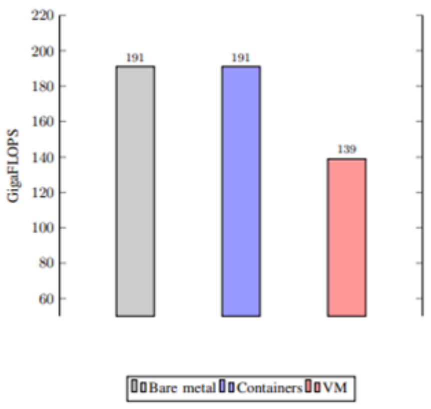


Figure 5: Linpack CPU Benchmarking

Multiple Machines (Cluster) with Multiple VMs and containers

a. Initial Human Efforts and Bootup Efficiency

15 graduates of a BigData course were asked to set up both clusters by following the guidelines in order to compare the efficiency of setting up a container Spark cluster with a virtual machine Spark cluster. Both the overall time and the amount of time spent on each step are noted. As a result, the amount of time they spent may mostly be attributed to the difficulty of configuring a container cluster and a virtual machine cluster [3]. Docker containers require 23 minutes to construct a cluster, whereas virtual machines (VMs) take 46 minutes, a 100% longer time. By studying the time spent by each of the three processes, we discover that containers save most of its time during the image building phase. One of the factors that contributes to this is installing the OS can be omitted because containers and the host machine share the same operating system. The other factor is that only one image file needs to be created for containers in order to set up the cluster because image files can be shared across other containers. However, because each virtual machine (VM) requires a different image to begin with, an image file must be duplicated several times during VM use. Using containers and virtual machines (VMs) takes about the same amount of time to set up Spark and launch the cluster. This led us to the conclusion that a cluster of containers are much easier to set up than a cluster of VMs. By changing the number of instances that boot up, the figure compares the bootup delay of virtual machines and Docker containers. The amount of time that passes between the first instance starting up and the last instance finishing up booting up in each set of trials were measured [3]. All instances are started sequentially.
A few noteworthy findings are as follows: on the one hand, the real computer has a greater capacity for Docker containers than virtual machines. We concluded that when there are about 250 idle virtual machines on the host, it is really challenging to start new ones. For instance, the actual machine responds very slowly and is nearly unusable, while it takes over 1000 seconds to boot up a single virtual machine. Conversely, 512 Docker containers may be started in just 987 seconds, or

1.89 seconds on average per container. However, Docker containers require far less time than virtual machines (VMs) to launch an equivalent number of instances, particularly when there are many instances. We can observe that the bootup delay of two Docker containers or virtual machines is nearly the same. But when this figure rises to 256, the VM takes 24295 seconds, 50.72 times longer, and Docker containers take 479 seconds. This figure makes a number of intriguing findings.
First, less memory is used at startup by the Docker container and the virtual machine. Although each instance is permitted to use up to 2GB of RAM, a virtual machine typically needs 0.23GB of memory after booting up. This method reduces the amount of idle resources in each instance by allocating resources to a virtual machine (VM) or Docker container only when necessary. Second, after booting up, a Docker container uses less memory than a virtual machine. Tests indicate that a Docker container uses only 0.03GB of RAM, compared to 0.23GB for a virtual machine. This is due to the fact that a container can share an operating system with the physical computer, whereas a virtual machine must maintain its own operating system. This explains why more containers than virtual machines can be stored on the same physical computer.

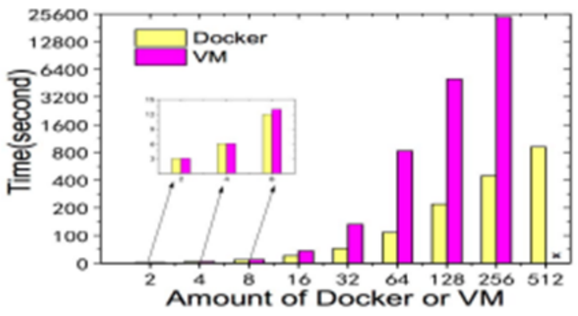


Figure 6: Bootup Latency (X means failed to boot up).

b. Execution Time

Several Spark workloads having emphasis on different types of resources were included to the test for the execution time. 4 machines are used as a slave and 1 machine is used as the master [3]. From the results that are shown in the figures, we can conclude that when the number of containers or virtual machines (VMs) rises, the execution time of each workload under evaluation first drops and subsequently increases. A reason for this is a Spark job typically comprises of several tasks that can be carried out concurrently in various containers or virtual machines, indicating a job with a certain number of tasks is run in more containers or virtual machines (VMs), the job's parallelism increases and its execution time decreases. Nevertheless, since each container or virtual machine must have access to hardware resources, the more containers or virtual machines there are on a single physical computer, the more intense the resource competition gets, which affects the containers' or virtual machines' performance. For this reason, adding more containers or virtual machines to the Spark cluster causes all four of the workloads to execute more slowly. The scalability of the two environments is also distinguished by the noticeable variation in the execution time of the same task when executing in virtual machines (VMs) and containers. First, compared to a virtual machine environment of the same size, the execution time of a single workload is shorter in a container environment. As a lightweight virtualization technique, containers, for instance, provide programs running inside of them with less runtime overhead than virtual machines (VMs), according to all of the comparison results. The reason is because the VM OS and the hypervisor layer both impose overhead for the programs in the form of extended network or disk I/O paths, privileged instruction translation, context switches, and similar processes like these. Second, containers offer a more scalable environment than virtual machines (VMs) as cluster sizes expand. The execution time difference between the virtual machine and container for logistic regression is only 2.76% when the cluster size is 4.

Nevertheless, Logistic Regression requires 238 seconds to complete when the cluster size increases to 44 in the container environment, whereas it takes 3643 seconds which is more than 15 times longer in the virtual machine environment. Additionally, the workload fails to execute when the cluster size reaches 48 or more because there are too many tasks that are abandoned in the virtual machine environment whereas it can still be completed effectively and within an acceptable amount of time in a container environment.

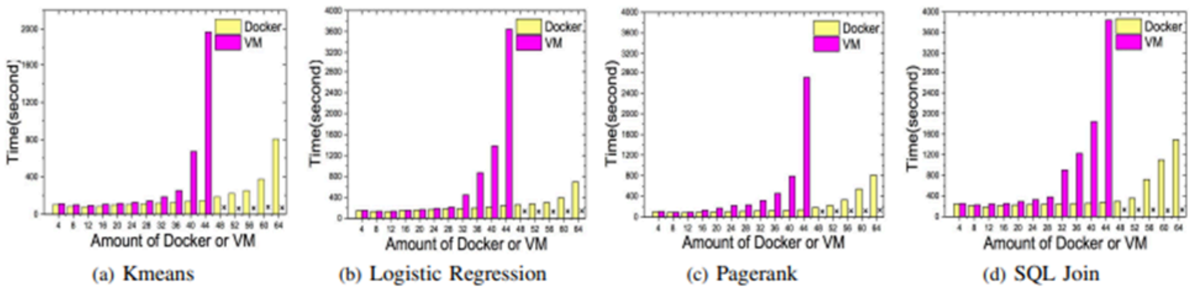


Figure : Execution Time Of Different Spark Workloads (X means the workload failed to finish).

c. CPU

As an effort to understand the reason why containers have a
much better scalability than VMs, CPU utilization on the task PageRank on each of the slave machines were collected and shown as a graph. The total CPU utilization is divided into three categories: user level CPU, system level CPU, and the wait time [3]. Because the PageRank workload is a user level application, and most of its instructions can run without being trapped into the kernel, we observe that the CPU is mostly spent on the user level and CPU wait, in both environments, in all the cluster sizes. A more important observation is that the CPU spent more time on wait with the increase of the cluster size, especially when VM is used. For instance, when the cluster size is 8, the average CPU wait time is less than 10% in both container and VM environment. On the other hand, when the cluster size increases to 48, the average CPU wait time in container environment is far less than that in the VM environment. In the VM environment, the amount of user level CPU that can be made use of by each task is much less than that in the container environment. A significant volume of disk I/O with lengthy latencies which is partially caused by memory swap, is also indicated by the high CPU wait time. Therefore, many tasks aborted due to timeout and PageRank failing in a 48 VMs cluster while being successfully finished in a 48 containers cluster validates this inference. Hence, these results led us to the conclusion that with the same workload, Dockers container achieves higher CPU utilization.

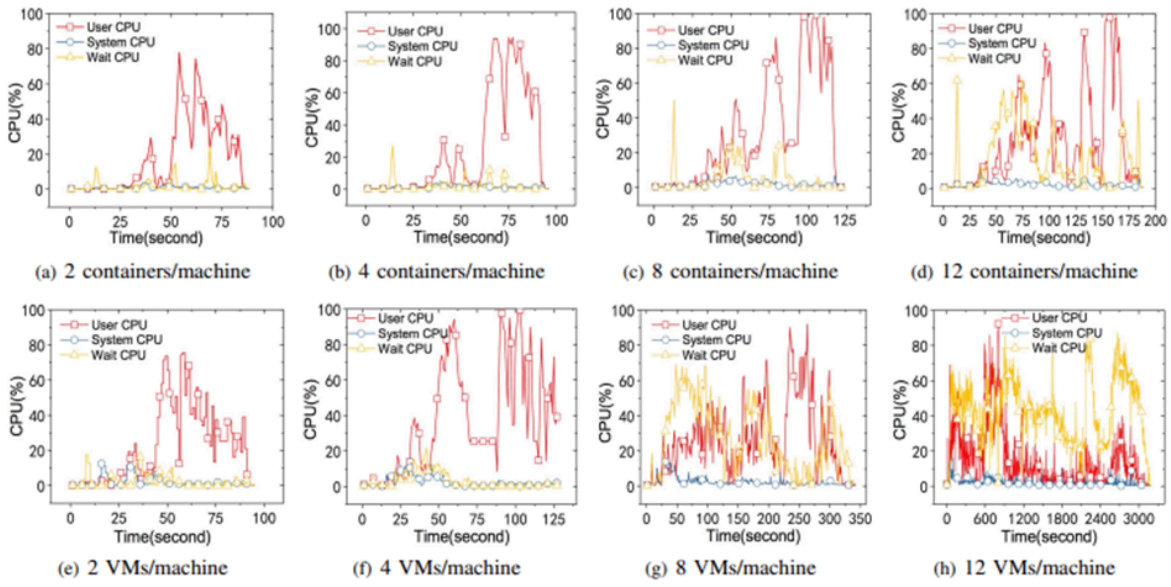


Figure: Average CPU utilization of all the slave machines with the PageRank.

d. Memory

Additionally gathered and displayed in the figures are the average memory usage statistics of the four slave machines throughout the PageRank execution [3]. A number of noteworthy observations are made.

First, even though each container and virtual machine (VM) has 2GB of RAM initially, two containers on each slave system use less than 2GB of memory, while two VMs require roughly 4GB of memory before the workload begins to execute. This is mostly because, unlike virtual machines (VMs), containers do not have to maintain an entire operating system. Second, compared to virtual machines, containers offer more flexibility in memory allocation. The findings demonstrate that a container's memory allocation starts off relatively tiny and subsequently grows in response to the needs of the applications running within it. However, a virtual machine initially uses more memory. Additionally, a virtual machine (VM) retains memory long after it falls idle, whereas a container releases memory after completing its workload. Therefore, based on these findings, we concluded that the Dockers container achieves higher memory utilization with the same workload.

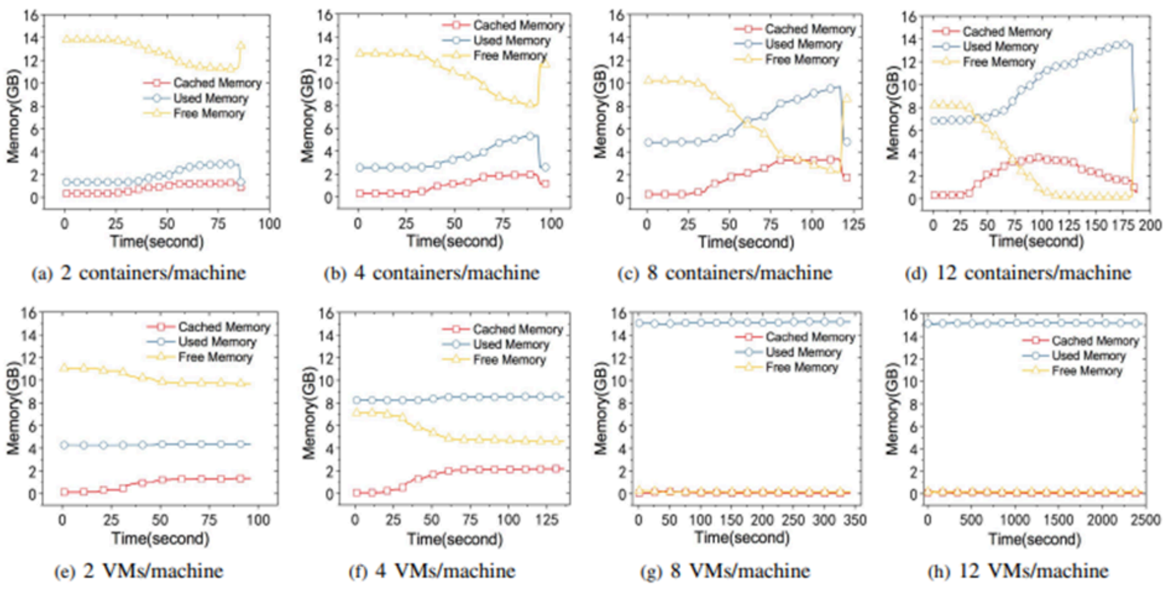


Figure: Average memory utilization of all the slave machines with the PageRank.

VMs and Containers in Data Centers

Comparing VMs and containers at scale requires thinking about more than just how they compare in terms of CPU, networking or other hardware-related factors. In data centers, what is more important is how these technologies can assist users and large companies to launch applications and ensure that there are safe techniques to achieve load-balancing, fault-tolerance and other mechanisms to ensure an infrastructure to rely on, as well as reasonable operational costs. In the upcoming three sections, we will discuss what the papers that we read got as results in terms of how the different characteristics of VMs and containers affect the various options for managing resources in a cluster.

a. Resource allocation

Data center and cloud operators use cluster management frameworks to meet application resource requirements. These frameworks rely on resource provisioning controls provided by hypervisors and operating systems to manage VMs and containers [1]. Hypervisors allocate virtual CPUs, memory, and I/O devices to VMs, enabling resource sharing at the hardware level. For containers, resource allocation includes additional operating system controls, such as CPU scheduling and memory swapping, alongside physical resource allocation, resulting in more granular provisioning [2].

Another important aspect of resource allocation is the fact that VMs and containers have different kinds of limits when it comes to utilizing resources. VMs operate with hard resource limits, meaning they cannot exceed their allocated resources, even if additional capacity is idle on the host [2]. Overcoming these limits is challenging because VMs are provisioned with fixed virtual hardware (e.g., CPUs, memory, I/O devices) during boot-up. Adjusting these allocations dynamically is extremely rare and impractical. In contrast, containers utilize soft limits, allowing them to exceed their assigned resources if excess capacity is available [2]. This flexibility promotes more efficient resource utilization, especially in overcommitted scenarios [2]. Overcommitment occurs when the total resource allocation exceeds the physical capacity of the host [2]. For example, if the host has 100 CPU units and the containers are collectively allocated 150 units, the overcommitment factor is 1.5. Containers can still function effectively in such environments because their soft limits enable them to adapt dynamically, using idle resources when needed without rigid restrictions. This adaptability makes containers highly efficient for managing resources in shared and variable workloads.

b. Migration

Migration is a key mechanism in data centers, used for tasks such as load balancing, fault tolerance, and resource consolidation. VM migration is a well-established, reliable, and widely adopted process [2]. It works by periodically copying memory pages from the source host to the destination, with management platforms governing the process through policies that account for resource availability on both hosts [2]. However, VM migration is resource-intensive as it involves transferring both the application state and the guest operating system state.[3]

In contrast, container migration is less mature and significantly more challenging [2]. It relies on process migration techniques, which must capture not only memory pages but also a large amount of associated operating system state, such as process control blocks, file tables, and sockets [2]. These dependencies require specialized libraries and kernel features, which may not be universally available, limiting viable destination hosts. In the meantime, killing and restarting stateless containers is a practical alternative for managing workloads. Consequently, container management frameworks do not yet support migration as reliably as VMs. However, containers offer advantages: their smaller memory footprint compared to VMs allows for potentially faster migrations [3].

c. Deployment

Launching applications with low latency is a critical feature of management frameworks, as it enables quick scaling to handle workload spikes. Furthermore, virtualization facilitates horizontal scaling by allowing multiple instances of an application to run simultaneously. Management frameworks support specifying the number of replicas for containers or VMs, which is particularly useful for managing load surges [2]. However, there is a notable difference in start-up times: virtual machines can take tens of seconds to boot, whereas containers can start in under a second, making them more suitable for rapid scaling [2].

On the other hand ,multi-tenancy, the ability to share a cluster among multiple users, is another important aspect of deployment. Virtual machines excel in multi-tenancy due to their strong resource isolation, which makes them inherently secure for sharing among untrusted users [2]. In contrast, containers have weaker isolation, which presents security risks in multi-tenant environments [2]. While container management platforms like Kubernetes are working on improving multi-tenancy support, safe deployment still requires extensive security configurations. Policies for secure container placement are needed to mitigate risks, as containers are not inherently "secure by default," unlike virtual machines.

Conclusions

As the results that were achieved by the papers that we chose are multi-dimensional, it is suitable to divide this part into three main sections:

a. Single Machine and Single VM/Container Performance
Containers consistently outperform virtual machines in bootup efficiency, disk I/O operations, network bandwidth, and CPU performance. Containers achieve these results due to their lightweight architecture, which avoids the overhead of a full guest operating system. The reduced startup times and minimal resource requirements of containers make them a preferable choice for applications demanding rapid initialization and efficient resource utilization.

b. Cluster-Level Performance
In cluster environments, containers demonstrate superior scalability and execution efficiency. They require significantly less time to configure and launch a cluster compared to virtual machines. Containers also handle resource competition more effectively, maintaining consistent performance and completing workloads that fail under VM-based clusters. These characteristics make containers more suitable for high-demand, resource-intensive Spark workloads in distributed computing environments.

c. Resource Management in Data Centers
In large-scale data centers, containers provide more granular and flexible resource allocation through soft limits, enabling dynamic adaptation to workload variations. However, virtual machines offer stronger isolation, making them a more secure option for multi-tenant environments. While container migration remains less mature and complex compared to VM migration, containers compensate with faster deployment times and lower resource overhead, enabling rapid scaling during workload surges.

Final Remarks
In summary, containers offer substantial advantages in terms of performance, scalability, and resource efficiency for most applications, particularly in single machine setups and scalable clusters. However, virtual machines remain indispensable in environments where strong security and multi-tenancy are critical. A hybrid approach, leveraging the strengths of both technologies, could provide the best balance for diverse workloads and infrastructure needs.

References

[1] *Containers vs. virtual machines*, **Ian Buchanan**. Atlassian (2024). https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms (accessed 2024-11-20).

[2] *Containers and Virtual Machines at Scale: A Comparative Study*, **P. Sharma, L. Chaufournier, P. Shenoy and Y.C. Tay**. Association for Computing Machinery (2016). https://dl.acm.org/doi/10.1145/2988336.2988337 (accessed 2024-11-20).

[3] *A Comparative Study of Containers and Virtual Machines in Big Data Environment*, **Q. Zhang, L. Liu, C. Pu, Q. Duo, L. Wu, W. Zhou**. (July 2018). https://arxiv.org/abs/1807.01842 (accessed 2024-11-20).

[4] *Containers & Virtual machines: A performance, resource & power consumption comparison*, **Martin Lindström**. Blekinge Institute of Technology (June 2022). https://www.diva-portal.org/smash/get/diva2:1665606/FULLTEXT01.pdf (accessed 2024-11-21).

[5] *Comparative Study of Virtual Machines and Containers for DevOps Developers*, **S. Deochake, S. Maheshwari, R. De, A. Grover**. Rutgers University (April 2023). https://arxiv.org/pdf/1808.08192 (accessed 2024-11-22).