Cloud Computing Report

Lorenzo Cusin - Università degli studi di Trieste

21 February 2025

Abstract

This report compares the performance of Virtual Machines (VMs) and Containers (C) on an Apple MacBook Air M1, analyzing their efficiency across multiple computing aspects. Virtualization plays a crucial role in modern computing, with containers gaining popularity due to their lightweight nature. To evaluate performance, a series of benchmark tests were conducted, including HPL (CPU performance), Sysbench (CPU and memory), IOzone (I/O and NFS), and Iperf (network throughput). The empirical results confirm theoretical expectations, demonstrating that containers consistently outperform virtual machines in CPU efficiency, memory management, disk I/O, and network performance. The reduced overhead in containers contributes to superior performance compared to VMs, which suffer from additional abstraction layers and emulation overhead. This study highlights the advantages of container-based virtualization for high-performance computing and application deployment. All the project components, like scripts and logs, can be found in the GitHub repo (`https://github.com/Lowry02/cloud.git`)

1 Introduction

In this report, the performance of Virtual Machines (VM)[1] and Containers (C)[2] will be compared. Both represent widely used methods for performing virtualization/app isolation in host hardware, with the latter growing in popularity [3]. This report aims to verify whether the theoretical knowledge of this field, which will be briefly presented in the following chapters, will be verified by empirical results. To obtain a general overview of performance, I will do the following test:

1. **HPL:** *CPU* test;
2. **Sysbench:** *CPU* and *MEMORY* test;
3. **IOzone:** *I/O* test and *NFS* test;
4. **Iperf:** *NETWORK* test.

A Macbook Air M1 will be used as host hardware. To avoid any ambiguity, its specifications are listed:

- **CPU**: Apple M1 chip, 8-core CPU with 4 performance cores and 4 efficiency cores;
- **MEMORY**: 8GB unified memory (RAM);
- **Storage**: 512GB SSD.

Finally, significant attention will be paid to the workflow pipeline and it will be accurately described.

2 Preliminary knowledge

Virtual Machines and Containers are both technologies used for application isolation, but they operate differently in terms of resource utilization and architecture. A Virtual Machine is a fully isolated environment that runs its operating system on top of a host system (managed by a **hypervisor**). Each VM has its own OS, kernel, and dependencies, making it resource-intensive but offering strong isolation. Containers, instead, provide application isolation by sharing the host OS kernel while encapsulating the application and its dependencies. Instead of using a hypervisor, containers rely on a container runtime. This makes Containers more lightweight and faster to start compared to VMs.

Given this knowledge, it is important to specify that macOS does not natively support running VMs or Containers. So it needs software like VirtualBox (for VM) and Docker (for C). Moreover, Docker and Containers, in general, do not need a virtualization because they leverage on some important Linux kernel features like namespaces, so this makes it more efficient in most scenarios. Unfortunately, macOS does not implement a Linux Kernel natively, so Containers need virtualization but it is more lightweight than Virtual Machines one.

So, considering the theoretical aspects, I expect Containers to have better performance than Virtual Machines. Let's discover it.

3 Pipeline and test setup

Before starting, it's important to notice that some benchmarks, in particular the ones related to the *CPU* and *MEMORY*, are subject to fluctuations due to the hidden processes that the host is running. So, I decided to define a workflow that gave me the possibility to evaluate that variance in order to obtain a robust estimation of the metrics. Moreover, the execution time was also something to take into account, so the final pipeline represents a trade-off between

the need for a robust estimation and a non-extreme time-consuming setting.

I decided to perform several executions (RUN) of the same benchmark. I initially set $RUNS = 5$ to have an idea of the overall variance and, if it were too high to draw conclusions, I would increase it.

I also created some configuration and execution scripts: the first one was used to configure every single VMs and Cs, and the second one was to perform tests and save logs. This facilitated the entire workflow.

To configure Docker containers, I decided to leverage on Dockerfile, in order to create a Docker image that incorporates the setup scripts previously mentioned and Docker-compose, used to set up the entire infrastructure, with resource limitations and network configurations.

3.1 Setup

VirtualBox[4] and Docker[5] were used to create the two types of virtualization and I created an identical configuration for the Virtual Machines and Containers, let's call them respectively:

- **Large configuration**: 4CPU, 4GB RAM, 25GB disk, IP: 192.168.56.10 (associated to vm1 (c1) in */etc/hosts* file);
- **Small configuration**: 2CPU, 2GB RAM, 25GB disk, IP: 192.168.56.11 (associated to vm2 (c2) in */etc/hosts* file).

An Ubuntu Server 24.04.1[6] was installed in both and static IPs were set. The IPs were also associated with a hostname to facilitate their use. For the VMs, the network had two adapters: the first one to permit communication between them and the second one to permit access to the internet; in containers, the default configuration was used.

All the tests ran in an environment in which only the VM (C) involved was executed. Only in the case in which both of them were needed for the test (like for the network test), both configurations ran simultaneously.

Finally, in order to obtain the maximum performance from Docker, I disabled the Rosetta x86_-64/amd64 emulation (MacBook Air M1 has an ARM architecture).

4 HPL Benchmark

The High-Performance Linpack (HPL)[7] benchmark is a software package used to measure the floating-point performance of computers. It solves a dense linear system of equations and returns the Floating point Operations Per Second (FLOPS) performed. It has several tunable parameters, the remarkable ones are:

- **N**: size of the side of the matrix; N^2 represents the number of values in the matrix (saved in double precision format);
- **NB**: Blocks size used to divide the matrix problem into sub-problems (granularity parameter). It is recommended a value between 32 and 256 (given by empirical results)[8][9];
- **P and Q**: $P \times Q$ processes matrix to distribute workload; in general square-like grid is preferable.[9]

I decided to try different configurations of N and NB to assess their impact on performance. For P and Q selection, instead, I used a different approach that involved merging two constraints: $P \times Q = number\ of\ cores$ and $P \approx Q$.

So, for the *large* and *small* configurations introduced in 3.1, I selected all the combinations given by:

	N	NB	P	Q
Large	1024	32	2	2
	2048	64		
	4096	128		
	8192	254		
Small	1024	32	1	1
	2048	64		
	4096	128	2	2
	8192	254		

Let's notice that for the *small* configuration, it wasn't possible to obtain a square-like process grid, so I searched for the optimal set for P and Q.

The tests ran using core binding, in this way each process was exclusively using one single core. This choice reduced the latency and increased performance. Moreover, Docker gives the possibility to do core-pinning: I also investigated this setting.

4.1 Result

As previously stated, I initially investigated if the core-binding setting, which is a simple flag set on the command used to execute the HPL test, could be useful in improving performance. I'm not reporting the data here, but the improvement was largely evident. So, in all the VMs and Cs configurations, I used it.

Let's analyze the different performance for the *small* configuration given by different P and Q values:

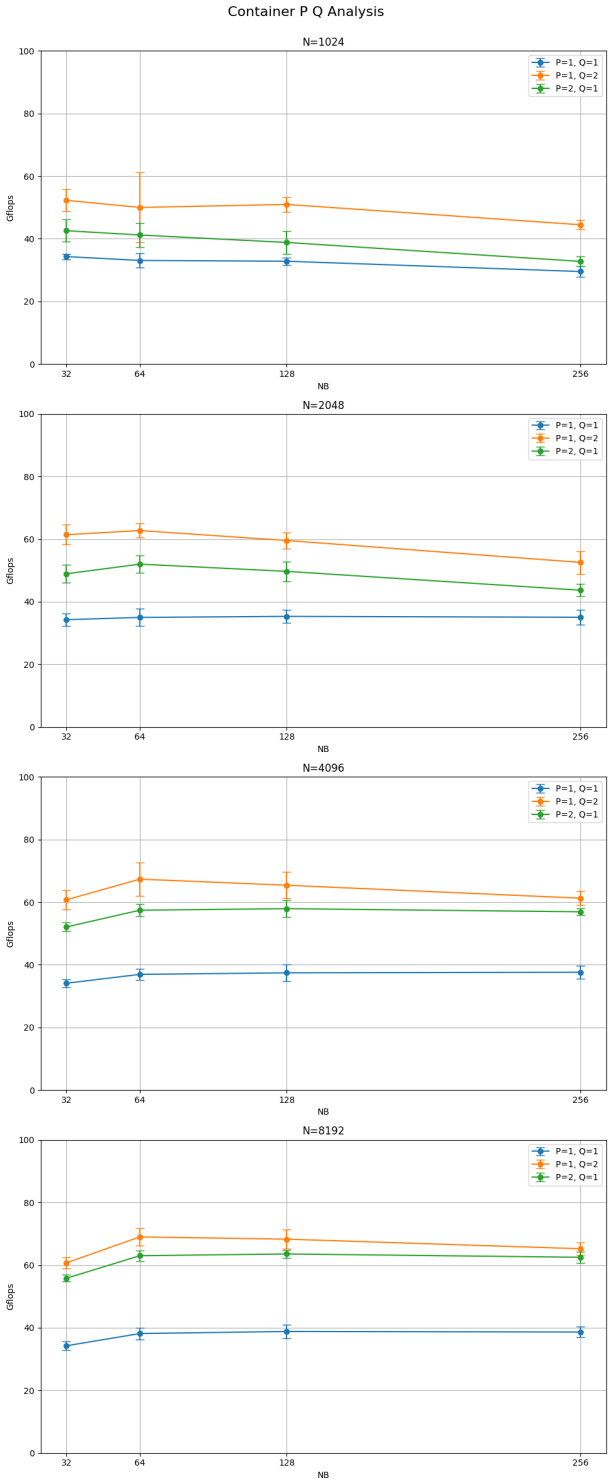

Figure 1: Container P and Q Analysis

Here I reported only the C results, from the VM ones the same conclusions can be drawn. It is visible that, independently on the N value, $P = 1$ and $Q = 2$ are in general the best setting. I also reported the case with $P = Q = 1$ to give an idea of the difference with a single process test.

For the next analyses, I will only consider the results with the optimal P and Q.

As mentioned in the section 4, Docker gives the core pinning possibility. This means that the Container can only access a specific set of cores. Knowing that M1 Apple Silicon has cores dedicated to different scopes (performance and efficiency), the analyses of the difference could be interesting:

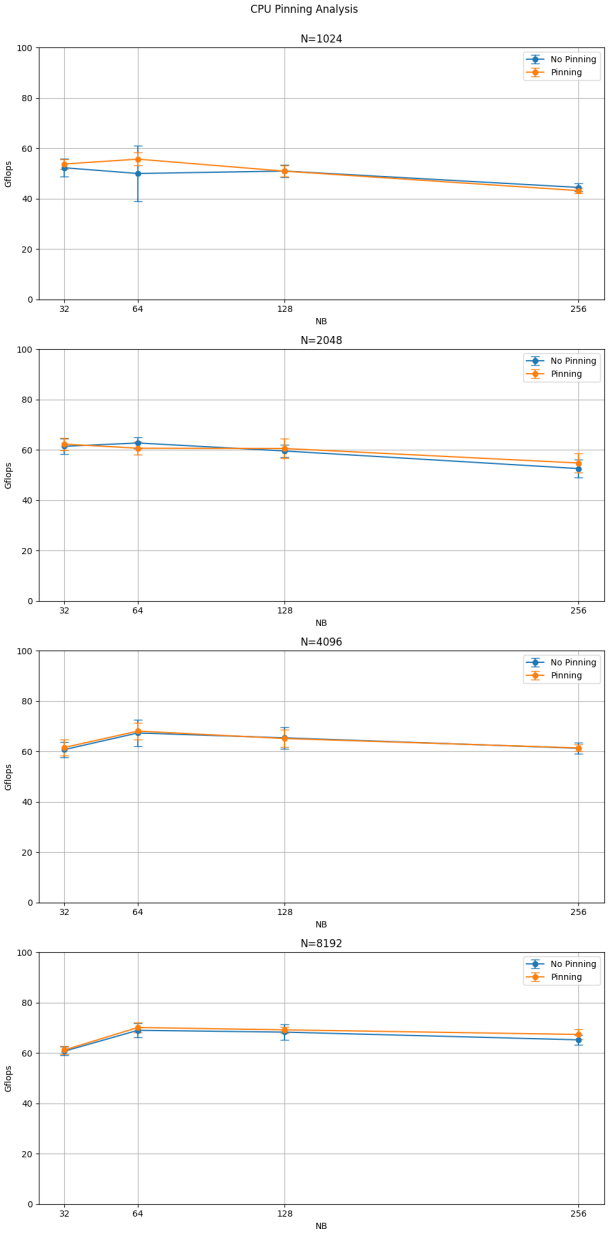

Figure 2: *Small* container CPU pinning

The *small* container results are shown, but the same behavior is present in the *large* one. We can notice that there is no significant difference between the two settings, with a small advantage for the 'no-pinning' configuration. I decided to consider the latter as the best one in order to give the system wider freedom in the management of the resources.

Let's now compare the VMs and Cs performance:

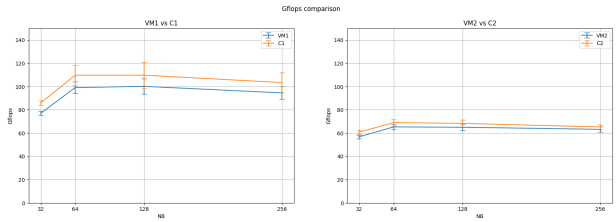

Figure 3: VMs and Cs Gflops comparison

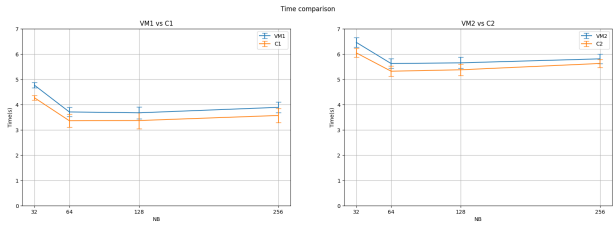

Figure 4: VMs and Cs Time comparison

First of all, we can see, as we could imagine, that the *large* configuration (VM1 and C1) has better performance than the *small* configuration (VM2 and C2), having more resources. But, most importantly, the results we were expecting from the theoretical point of view are respected: the Cs are performing better than VMs ($\sim$ 10% difference in terms of Gflops).

It was in my interest to compare the obtained results with the real hardware performance but they could be subject to some biases. In fact, to compile the HPL benchmark for the M1 Apple Silicon Chip, we need to change the configuration of the compiler to adapt to the ARM architecture and the specifications of the CPU. This process could involve particular optimization leading to different executable files: so in the virtualization environment we would have a different setup than the macOS one. However, I want still to mention that it achieved $\sim 180\ Gflops$ in the best case, but I also want to warn the reader that the comparison with the virtualization results could be misleading.

I was still curious to know if the virtualization was close to the native performance of the hardware. So I decided to evaluate the theoretical performance of the CPU, the one given by:

$$GFLOPS = cores \times clock(GHz) \times \frac{FLOPs}{cycle}$$

Considering the *large* configuration (4 cores), I ended up with the following value:

$$GFLOPS = 4 \times 3.2GHz \times 8 = 102.4$$

This value is lower than the ones obtained by the tests. It could seem strange but it is not so unusual: M1 has a big.LITTLE architecture that combines high performance with high-efficiency cores, that could lead to some discrepancy concerning the theoretical evaluation.

5 Sysbench Benchmark

Sysbench[10] is an open-source benchmarking tool used to evaluate system performance, including CPU, memory, disk I/O, and database workloads. In this report, it was used to benchmark both CPU and MEMORY performance. CPU performance is evaluated through a series of prime number calculations, effectively stressing the processor's integer operation capabilities. Meanwhile, memory performance is assessed through a series of sequential read and write operations. The remarkable metrics that it returns are:

1. **CPU events/s**: number of computational tasks (or events) the CPU completes per second during the benchmark test. A higher value indicates better CPU performance, as the processor is handling more operations in less time;

2. **CPU latency (ms)**: time taken (in milliseconds) to complete an individual computational event. Lower latency means the CPU processes tasks more quickly, which is desirable for high-performance systems;

3. **MEMORY total time (s)**: total duration (in seconds) taken to complete the memory benchmark test. It gives an overall measure of how long the memory read/write operations took during the test;

4. **MEMORY latency (ms)** – time taken (in milliseconds) for a single memory operation (read or write) to be executed. Lower memory latency means faster memory access, which improves overall system performance.

Let's initially compare the performance between the *small* and *large* configurations:

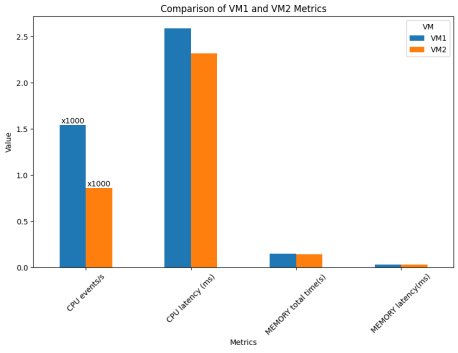

Figure 5: Sysbench VMs comparison

We can see that in general, as previously said, the performance is better for the *large* configuration in terms of CPU performance. We cannot state the

same for the *MEMORY*, or at least the differences are not so evident. The same conclusions can be drawn for the Cs.

Let's now compare VMs and Cs:

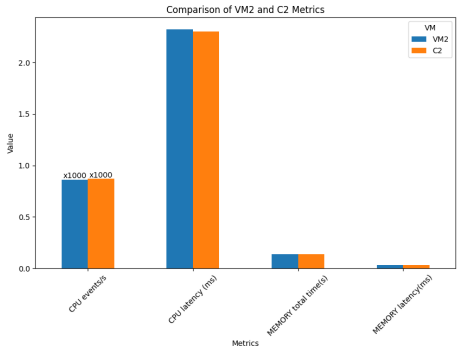

Figure 6: Sysbench *Large* configuration comparison

The CPU conclusions are more or less the same as the HPL benchmark: the Cs perform slightly better than VMs (same for the *small* configuration). A better latency gives a good insight into the difference in terms of Gflops (seen in 4) and CPU Events metric between the two configurations. There are no meaningful differences in terms of MEMORY latency and events.

6 IOzone Benchmark

IOzone[11] is a widely used benchmarking tool that evaluates the performance of a system's file I/O operations, including read, write, re-read, re-write, random access, and more. It provides insights into how efficiently a system handles disk operations, making it useful for analyzing storage performance under different workloads. In this report I will consider only the *write* and *read* metrics:

- **Write Test**: generates a file and writes data sequentially.
- **Read Test**: reads the file sequentially to gauge read performance.

The test produces metrics based on different *File sizes* and *Record Length* (size of the read block).

6.1 Write test

First of all, let's compare the results between different configurations:

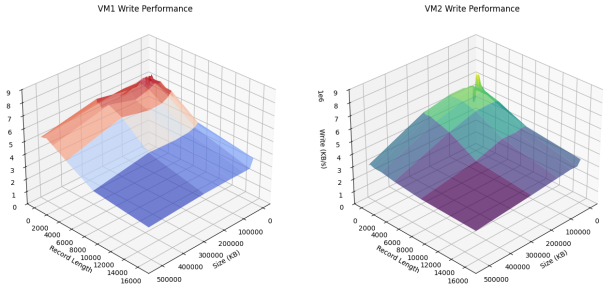

Figure 7: IOzone VMs write test comparison

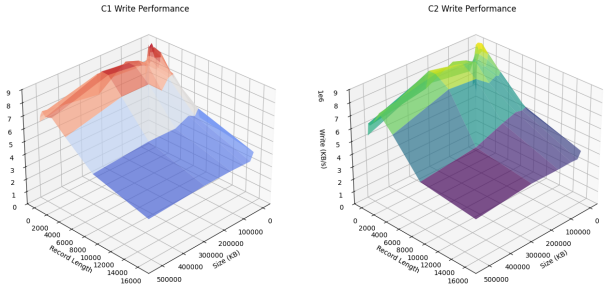

Figure 8: IOzone Cs write test comparison

As we can see, the VMs present an evident difference. We can notice that, for large *File sizes*, the *large* configuration is able to manage in a better way the smaller Record Length values. This behavior is also present in the Cs but it is not so marked. Probably this difference is due to the lightweight virtualization that Cs have.

6.2 Read test

Let's compare the two configurations:

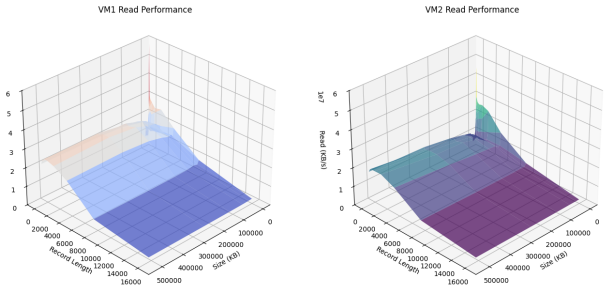

Figure 9: IOzone VMs read test comparison

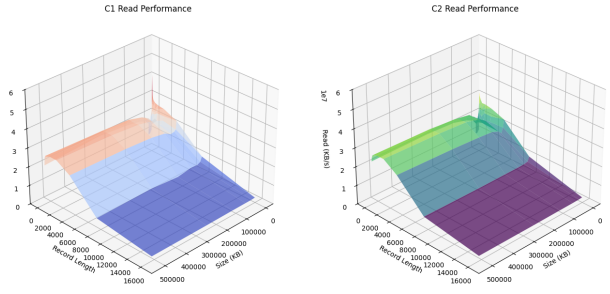

Figure 10: IOzone Cs read test comparison

In this case, the shape of the surface, which represents how the VM/C is able to manage the workload with different settings, is more or less the same. The only difference is about the amount of *KB/s*, due to the different number of cores.

6.3 VMs and Cs comaprison

I will compare VMs and Cs using the following approach: given a specific *File Size*, I will keep only the best performance obtained in terms of *KB/s*, independently from the *Record Length*. Here are the results:

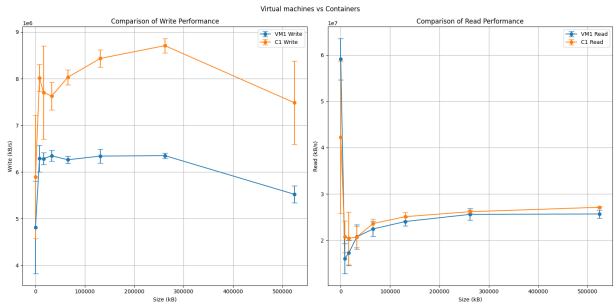

Figure 11: IOzone comparison between VMs and Cs

Notably, the writing performance is in favor of the Cs, instead, the read performance is nearly close to each other. The virtualization latency/overhead produced by VMs is strongly influencing the obtained metrics here too. Reading, instead, seems to be well-optimized.

6.4 NFS test

A lot of times, VMs share disk resources through the network (NFS). It is in the interest of this report to analyze the relative performance. I configured an NFS using the *small* configuration VM as a server, and the *large* one as a host. I ran the test on the latter.

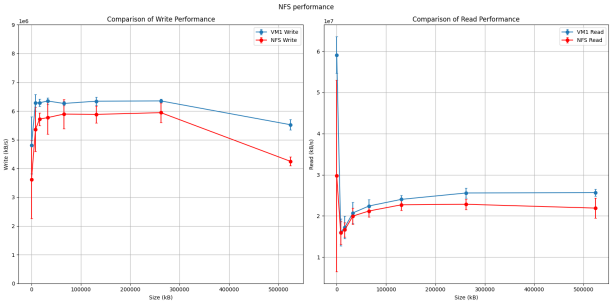

Figure 12: IOzone NFS performance

As we could expect, the performance is slightly worse than the one previously obtained.

7 Iperf Benchmark

Iperf[12] is a network performance measurement tool used to analyze bandwidth. It operates as a client-server application, generating TCP traffic to evaluate network throughput. It provides real-time bandwidth statistics at each second, allowing users to monitor fluctuations. Here are the considered metrics:

- **Bandwidth**: maximum data transfer rate of a network connection;
- **Transferred Data**: amount of data transferred during the test.

Here are the obtained results:

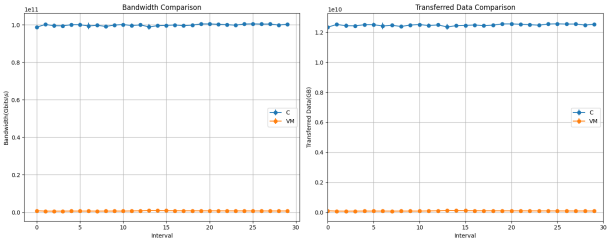

Figure 13: Iperf performance

In general, the Docker Network system is widely more efficient than the VirtualBox one. Both *Bandwidth* and *Transferred Data* are in favor of Cs. This result is aligned with the trend seen in this report.

8 Problems

The conduct of this project was subject to some problems and difficulties. Most of them were related to understanding how Docker and VirtualBox work in a M1 Apple Silicon setting. For me, it was not a trivial task, due to the black box characteristic of the Apple environment. Several strange behaviors (like in

the installation of VirtualBox Guest Additions) appeared during the development of the project. Probably working on a Linux environment would have been easier.

9 Possible improvements

I think that the results obtained are valid but they can be strengthened using a larger sample size(> 5 which was used in this paper). Also, investigations about different Docker virtualization strategies could be analyzed, like Docker VMM and QEMU [13], to understand their impact on the performance. The creation of a solid and robust pipeline to compare the virtualization results with the native ones could also be of interest. Finally, replicating the work done in this report in a Linux environment (in which containerization is native) could also give a better overview of the difference between the two isolation methods.

10 Conclusion

In this report, I explored the differences between Virtualization/isolation methods from a theoretical and empirical point of view. The main purpose was to demonstrate that Containers can give better performance than Virtual Machines, justifying the nowadays wide use. The pipeline was clear and tried to acquire as many robust conclusions as possible, implementing different runs of the same test to avoid misleading findings given by some variance factors. From what is stated above, we can affirm that Containers are an efficient and easy-to-use alternative to Virtual Machines in the context of app isolation.

References

[1] Ibm. *Virtual Machines*. Sept. 2, 2024. URL: https://www.ibm.com/it-it/topics/virtual-machines (visited on 02/15/2025).

[2] Ibm. *containers*. Aug. 27, 2024. URL: https://www.ibm.com/it-it/topics/containers (visited on 02/15/2025).

[3] Ibm Cloud Team. *Containers Vs VMS*. Nov. 25, 2024. URL: https://www.ibm.com/think/topics/containers-vs-vms (visited on 02/15/2025).

[4] *Chapter 6. Virtual Networking*. URL: https://www.virtualbox.org/manual/ch06.html (visited on 02/15/2025).

[5] Simeon Ratliff. *Docker: Accelerated Container Application Development*. Jan. 23, 2025. URL: https://www.docker.com/ (visited on 02/15/2025).

[6] *Get Ubuntu Server — Download — Ubuntu*. URL: https://ubuntu.com/download/server (visited on 02/15/2025).

[7] *HPL Tuning*. URL: https://www.netlib.org/benchmark/hpl/tuning.html (visited on 02/15/2025).

[8] *Profiling and Tuning Linpack: A Step-by-Step Guide*. Oct. 21, 2015. URL: https://community.arm.com/arm-community-blogs/b/servers-and-cloud-computing-blog/posts/profiling-and-tuning-linpack-step-step-guide (visited on 02/15/2025).

[9] Open-Power. *op-benchmark-recipes/standard-benchmarks/HPL/Linpack$_H$PL.dat$_t$uning.mdatmasteropen−power/op−benchmark−recipes*. URL: https://github.com/open-power/op-benchmark-recipes/blob/master/standard-benchmarks/HPL/Linpack_HPL.dat_tuning.md (visited on 02/15/2025).

[10] Akopytov. *GitHub - akopytov/sysbench: Scriptable database and system performance benchmark*. URL: https://github.com/akopytov/sysbench (visited on 02/15/2025).

[11] *IoZone Filesystem Benchmark*. URL: https://www.iozone.org/ (visited on 02/15/2025).

[12] Vivien Gueant. *iPerf - iPerf3 and iPerf2 user documentation*. URL: https://iperf.fr/iperf-doc.php (visited on 02/15/2025).

[13] *"Virtual Machine Manager"*. Jan. 21, 2025. URL: https://docs.docker.com/desktop/features/vmm/ (visited on 02/15/2025).