Cloud Computing Performance Testing

Giacomo Serafini - Università degli Studi di Trieste

21 February 2025

1 Introduction & Set-Up

This report evaluates the performance of virtual machines (VMs) and containers (CTs) using a suite of benchmarks, including `High Performance Linpack (HPL)` and `sysbench` for CPU and memory tests, `iozone` for disk I/O, and `iperf3` for network performance. The experiments were conducted on a host machine with the following specifications:

- **OS**: Windows 11
- **CPU**: Intel Core i5-10600K (6 cores/12 threads, Base/Boost clock: 4.10/4.80 GHz)
- **RAM**: 16GB LPDDR4 @ 3200MHz
- **Storage**: Sabrent 1TB NVMe

For all tests, all unnecessary tasks and applications were closed to ensure the best possible performance, leaving only the VMs or CTs active. To improve the reliability of the results, each test was repeated 5 times and the average was computed to smooth out fluctuations in the system performance. All performance logs and relevant data are available on the GitHub repository[1].

1.1 Virtual Machines (VMs)

Two virtual machines were set up in VirtualBox[2] using the `ubuntu-24.10-live-server-amd64.iso`. The first, named "Master", was configured with 4 cores, 4GB of RAM, and 20GB of storage. The second, "VM01", was provisioned with 2 cores, 2GB of RAM, and 15GB of storage.

Each VM was equipped with two network adapters: Adapter 1 was set to NAT, while Adapter 2 was configured as Host-only to facilitate direct communication between the two machines. This setup resulted in the VMs receiving the IP addresses 192.168.56.101 and 192.168.56.102, respectively. Additionally, SSH was enabled to allow remote access to both VMs from the host.

1.2 Containers (CTs)

Two Docker containers[3] were deployed using a `docker-compose` configuration. The first container, named "container1", was configured with a CPU pinned to cores `0,2,4,6` and allocated 4GB of memory. The second container, "container2", was set up with a CPU pinned to cores `8,10` and was allocated 2GB of memory. Both containers were built from the Dockerfile located in the `./app` directory and shared an internal bridge network (`internal_net`) to enable inter-container communication. Each container operated in privileged mode with the `SYS_NICE` capability added to improve CPU scheduling. The Dockerfile is based on `ubuntu:latest` and installs all necessary dependencies and benchmarking tools such as `iozone3`, `iperf3`, `sysbench`, and additional utilities required for performance evaluations.

Again, both the docker-compose.yml and the Dockerfile are available on the repository.

2 HPL

`High Performance Linpack (HPL)`[4] is a benchmark used to measure the floating-point computing performance of a system by solving a dense system of linear equations. In this evaluation, `HPL` was executed on both virtual machines and containers to assess their computational efficiency under numerically intensive workloads. Each test was repeated multiple times, with the average performance recorded to mitigate transient variations. The resulting metrics, expressed in GFlops, can be seen in the following Figure 1, enabling a detailed comparison of the performance differences between VMs and CTs.

Before showing the results, it is important to talk about the benchmark-tuning process. Although the `HPL` benchmark is part of the `hpcc` package, during container testing it caused major problems in performance output. In fact, the first runs of the containers showed results that were 10 times worse than what were later obtained. For this reason, it had been mandatory to build the whole benchmark from scratch, thanks to the work outlined in this blog.

However, being able to build it from scratch made it possible to tune it for the system. Following the official guidelines [5][6][7], many problem sizes N were tested, together with different block sizes NB. The tests, which can be seen in the repository, outlined $N = 9984$ for the smaller VM and CT and $N = 17664$ for the larger ones as the best performing problem sizes. Keeping these Ns

constant, the VMs and the CTs were tested on 5 different block sizes, from 32 to 256 as stated in the docs. Both HPL.dat files are located in the corresponding folders: the one for CT1 and VM Master, and the one for CT2 and VM01.

These are the results obtained from the tests:

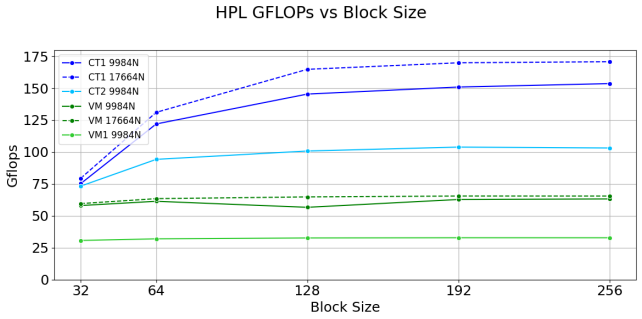

Figure 1: `HPL` GFLOPs performance comparison

In Figure 1, it is evident that containers consistently outperform virtual machines across all tested block sizes. In particular, the container configured with 4 CPU cores (CT1) achieves the highest GFLOPs values, with a clear margin over the other configurations. Even the container with 2 CPU cores (CT2) demonstrates better performance than both virtual machines at comparable matrix sizes.

This performance gap highlights the reduced overhead in container-based virtualization, compared to the more isolated environment of virtual machines. Consequently, for numerically intensive workloads like `HPL`, containers can harness system resources more efficiently than VMs, leading to higher sustained throughput in floating-point computations. This benchmark offered more information about the performance, all of which are stored in the repository. Still, all of them showed the Containers as the clear winners.

GFLOPs: Although a first look at these results can already tell much, it is important to compare these results to the theoretical ones, which can be obtained as follow:

$$no.\ cores \times Clock\ Speed\ (GHz) \times FLOPs\ per\ Cycle,$$

which produces 131.2 theoretical GFLOPs for the smaller VM and CT, and 262.4 GFLOPs for the larger ones.

Comparing these results with what was previously obtained, we can see that containers come closer to those theoretical values, being still far from close. The highest value obtained from `Container1` with $N = 17664$ and $NB = 256$ was 170.7, which is approximately 35% lower compared to the theoretical one. On the other hand, `Container2` got 103.83 GFLOPs, which is 20% smaller than 131.2.

A key reason for the better performance of containers over virtual machines lies in their lower overhead. Containers share the host operating system kernel and therefore require fewer layers of abstraction. In contrast, virtual machines must run on top of a hypervisor and maintain their own guest operating systems with their own full kernel and system services, leading to significantly higher overhead. This streamlined approach in containers results in more efficient CPU utilization and, consequently, higher sustained GFLOPs for computationally heavy workloads like `HPL`. Even when running on Windows 11, Docker containers typically still exhibit lower overhead, as this extra layer of virtualization is highly optimized for running container workloads.

Nonetheless, even containers do not achieve the theoretical maximums because real-world conditions always introduce additional overhead. Additionally, the theoretical formula assumes a best-case scenario, one in which floating-point units are continuously fed with data under perfect conditions.

3 Sysbench

Sysbench[8] is an open-source, multi-threaded benchmarking tool designed to evaluate various aspects of system performance, including CPU, memory, file I/O, and more. In this study, we focus on CPU and memory tests, running each benchmark for 60 seconds under two configurations (using 4 threads and 2 threads) to provide a comprehensive performance evaluation of virtual machines and containers. As for `HPL`, the benchmark offered more tests compared to what is displayed in this report, but all can be studied in the respective files stored on GitHub.

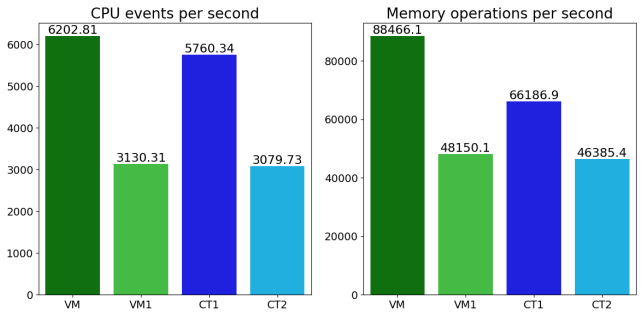

Figure 2: `sysbench` performance comparison

3.1 CPU Testing

For CPU performance, sysbench was executed with the following commands:

```
sysbench cpu --threads=4 --time=60 run
sysbench cpu --threads=2 --time=60 run
```

Referring to the left chart in Figure 2, the 4-core VM ("Master") achieves the highest number of CPU events per second, closely followed by the 4-core container (CT1). Meanwhile, the 2-core VM ("VM01") and 2-core container (CT2) exhibit proportionally lower throughput, with the VM slightly outperforming its container counterpart. This result indicates that, for CPU-bound tasks, VMs in this particular setup can deliver better performance than containers, although the difference diminishes with fewer allocated cores.

It is somewhat unexpected that, contrary to the results observed with `HPL`, the `sysbench` CPU tests show virtual machines outperforming containers. This discrepancy can be explained by the fact that, during the `HPL` benchmarking, the benchmark was finely tuned for the specific system, allowing the containers to reach their optimal performance. In contrast, `sysbench` does not offer similar tuning capabilities, leaving the performance output as is, thus the inherent overheads in containerization are more apparent in these tests.

3.2 Memory Testing

Memory performance was evaluated using sysbench's memory test with the following commands:

```
sysbench memory --threads=4
--memory-block-size=1M
--memory-total-size=10000G --time=60 run
sysbench memory --threads=2
--memory-block-size=1M
--memory-total-size=10000G --time=60 run
```

As shown in the right graph of Figure 2, memory throughput follows a similar pattern: the 4-core VM outperforms the 4-core container, while the 2-core VM leads the 2-core container by a smaller margin. The gap is more pronounced at higher core counts, suggesting that container-based memory operations may incur additional overhead compared to the more directly provisioned resources in a VM. Nevertheless, the containers' performance remains relatively close to that of the VMs, underscoring that, although the container abstraction layer can impact memory-intensive workloads, it does not drastically hinder overall throughput in typical `sysbench` scenarios.

As mentioned in the CPU Testing subsection, the lack of tuning options in `sysbench` means that the raw performance metrics are more directly influenced by the virtualization overhead. For this reason, virtual machines, with their dedicated resource allocation, exhibit a slight performance advantage over containers in memory-intensive tasks.

4 Iozone

The `iozone`[9] benchmark was used to assess disk I/O performance across both local storage and NFS environments. The tests were executed using the command:

```
iozone -I -a -g 512M -i 0 -i 1 -i 2
```

In this configuration, the `-I` option was used to disable the caching mechanisms of the operating system, thus providing a measure of the raw performance of the disk. It is noteworthy that running iozone without the `-I` flag yielded faster results due to the beneficial effects of caching.

4.1 Local Storage Testing

This subsection compares the disk performance of VMs and CTs using local storage benchmarks. Identical test conditions were maintained across both environments, and the performance metrics were recorded for analysis.

For clarity reasons, in this report only read and write operations will be compared, but all the other parts of the benchmark, as *Rewrite, Random write, Reread and Random read*, can be seen in the logs in the repository.

From the heatmaps in Figure 3, it is apparent that containers (top-right and bottom-right) generally achieve higher throughput and exhibit more uniform performance across a broad range of file and record sizes compared to virtual machines (top-left and bottom-left). While both VMs and CTs show improved read and write speeds as the file and record sizes increase, containers tend to maintain better consistency in throughput. This difference likely stems from the lighter-weight virtualization layer of containers, which reduces I/O overhead compared to the fully isolated environment of a VM.

Additionally, the performance gaps between VMs and CTs become more pronounced at larger file sizes, where the overhead of managing virtual disks in VMs begins to show. Containers, by contrast, are closer to bare-metal I/O performance because they do not require a dedicated virtual disk abstraction. Consequently, in local storage scenarios, containers often deliver faster read/write speeds than VMs.

In the following subsection, NFS performance will be examined, where network latency and protocol overhead can significantly affect overall I/O throughput.

4.2 NFS Testing

The NFS testing subsection focuses on evaluating disk I/O performance over a Network File System. The analysis provides insights into the behavior of VMs when accessing remote file systems through NFS. In this case, the bigger machine ("Master") acted as server, whereas the tests were performed on "MV01" to keep track of network overheads.

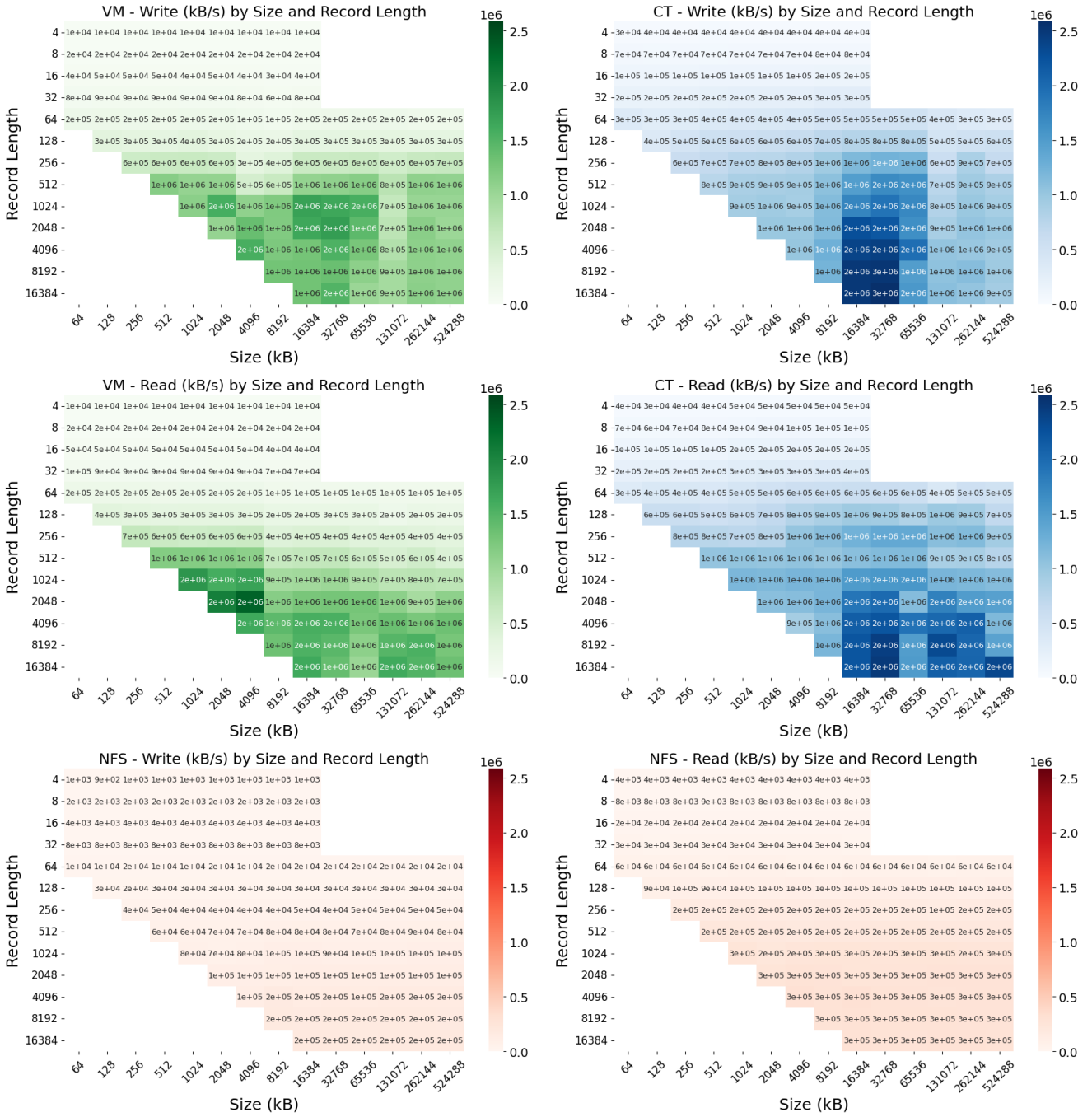

Figure 3: `iozone` read/write performance comparison

In the NFS tests (bottom panels of Figure 3), virtual machines exhibit lower read/write throughput than in local storage scenarios. This performance drop is primarily due to the additional overhead of network communication and potential latency spikes inherent to remote file system access.

5 Iperf3

`Iperf3`[10] is a tool that is used to measure network throughput, latency, and overall performance. In this study, `iperf3` tests were executed on both virtual machines (VMs) and containers (CTs) to evaluate their network performance. For each pair of tests, one system was first configured as the server while the other acted as the client. After completing the test in one direction, the roles were reversed to ensure a bidirectional assessment of network performance. This methodology

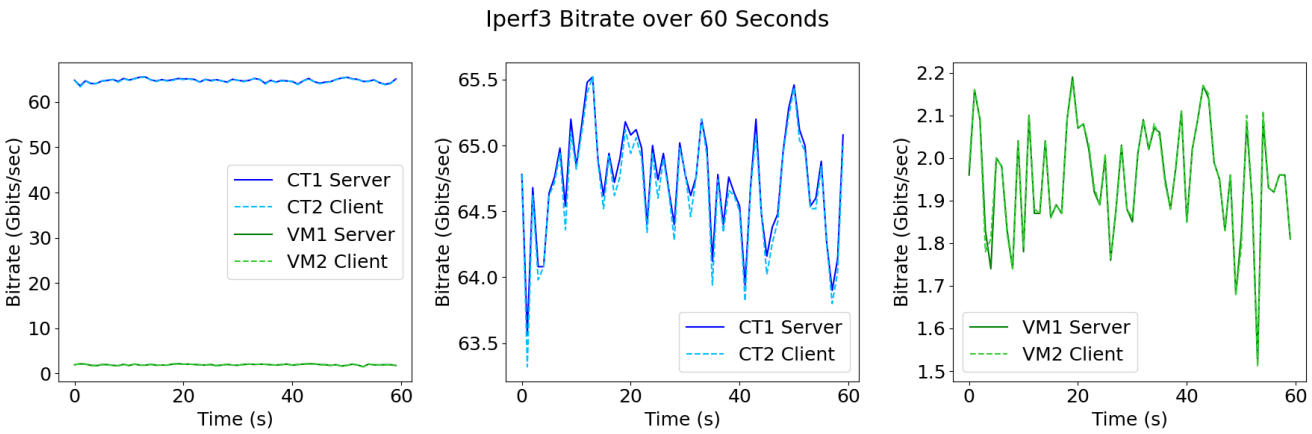

Figure 4: `iperf3` bitrate performance comparison

enabled a detailed comparison of network overhead and transmission efficiency between VMs and CTs. All these tests can be looked at in the repository, but to keep the report more readable only the case where the bigger machines acted as server is displayed.

From Figure 4, it is evident that containers achieve consistently higher throughput than virtual machines across the tested scenarios. In the left and center panels, where container-to-container communication is measured, the bitrate hovers around 60 Gbits/sec, despite some natural fluctuation. In contrast, the VM-to-VM transfer rate in the right panel peaks around 2 Gbits/sec, a substantially lower value.

The most likely explanation for this difference lies in the networking overhead introduced by virtual machines, which must route traffic through an additional virtualization layer and virtual NIC drivers. Containers, on the other hand, run closer to the host OS networking stack, thus incurring less overhead. Although both VMs and CTs are ultimately subject to the same physical network constraints, containers can leverage the underlying resources more efficiently, resulting in faster and more consistent throughput.

Conclusions

In this report, we performed a set of benchmarks comparing the performance of virtual machines and containers across various workloads, including computational tasks, disk I/O, and network throughput. Overall, our results indicate that containers generally deliver superior performance due to their lower overhead and more efficient use of host resources.

However, it is important to highlight that not all tests favored containers. For instance, in the sysbench CPU and memory evaluations, virtual machines showed a slight advantage. This outcome can be attributed to the fact that sysbench lacks the tuning capabilities available in benchmarks like HPL, where optimization for the specific system can significantly enhance container performance. Without such fine-tuning, the inherent overhead of containerization becomes more apparent.

In summary, while containers typically offer a more efficient virtualization solution, especially for workloads demanding high performance and low latency, the choice between containers and virtual machines should also consider specific application needs and security requirements to ensure optimal performance and resource utilization.

References

[1] *GitHub*. URL: https://github.com/.

[2] *VirtualBox*. URL: https://www.virtualbox.org/.

[3] *Docker*. URL: https://www.docker.com/.

[4] *HPL*. URL: https://www.netlib.org/benchmark/hpl/.

[5] *Tune HPL.dat*. URL: https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/.

[6] *HPL Tuning*. URL: https://www.netlib.org/benchmark/hpl/tuning.html.

[7] *HPL FAQ*. URL: https://www.netlib.org/benchmark/hpl/faqs.html.

[8] *sysbench*. URL: https://github.com/akopytov/sysbench.

[9] *IOzone*. URL: https://www.iozone.org/.

[10] *iPerf3*. URL: https://iperf.fr/.