# Container vs. Virtual Machines: An Survey on Latency Optimizations

Yiyang Xie, Florian Wiedner*, Jonas Andre*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: xieyi@in.tum.de, wiedner@net.in.tum.de, andre@net.in.tum.de

***Abstract*—With the launch of the URLLC (Ultra Reliable and Low Latency Communication) standard for 5G, low latency has become a fundamental requirement for almost all network systems. Traditional resource distribution and absolute latency values affect the service efficiency of computer networks, so virtualization and low latency are extremely essential for new services such as smart grid, driverless system and remote medical care. However, the combination of virtualization and low latency is quite challenging. In this work, virtualization and latency optimization will be covered. The focus is on containers and virtual machines, two of today's most advanced virtualization methods. Afterwards, different latency optimization techniques are explored. Finally, an analysis and outlook on the combination of the two is presented.**

***Index Terms*—containers, virtual machines, low latency, urllc, latency optimization**

## 1. Introduction

5G networks provide dedicated services for URLLC, and the requirements of the International Telecommunication Union (ITU) [1], [2] for URLLC services are defined as follows: the unidirectional delay from source to destination of a 5G radio access network (RAN) must not exceed $1ms$, and the delivery success rate must be higher than $99.999\%$. These requirements are extremely demanding for the systems that involve virtualization management. The virtualization of operating system (OS) allows multiple applications, in an environment sharing the identical host OS, to operate isolatedly. This has the advantages of fast bootup, easy deployment, low resource consumption and high operational efficiency [3], but also suffers from weak isolation, unstable communication connections and high network latency [4]. The latter prevents the successful performance of URLLC. This work summarizes the latest sophisticated virtualization methods and makes an examination of latency optimization methods in order to find virtualization solutions that support the URLLC service level.

The rest of this paper is structured as follows. In Section 2 the background is presented, while in Section 3 we focus on comparing container virtualization technology with virtual machine virtualization technology and summarizing the characteristics of the two OS virtualization technologies. The latency emerges as measurement latency and performance latency, both brought impacts should be avoided. In Section 4, it deals with optimizing the latency caused by the measurement process. The quantifying criteria and prerequisites for measurements are provided as well. In Section 5, the available and the potential performance latency optimization methods are overviewed. Finally, we obtain the conclusion in Section 6 and outline the challenges faced by virtualization techniques and the outlook for future research.

## 2. Background

Real-time systems have been gradually integrated into our life and future, from small real-time systems with soft deadlines such as e-games, video conferencing and virtual reality, to large real-time systems with hard deadlines such as industrial control systems and IoT-based Smart Cities (Internet of Thing) [5]. In these systems, especially hard real-time systems, the traditional physical hardware resources have been difficult to meet the computational demands during the program execution.

Virtualization technology has become a good solution for the drawbacks of high energy consumption, high maintenance cost and low flexibility of physical resource deployment. In brief, virtualization is the transformation of physical resources into logically manageable resources which breaks down the barriers between physical structures. In the context of 5G, resource sharing and optimization become particularly relevant. In addition, network performance and response time have become a non-negligible part of the puzzle. For hard real-time systems, there is little tolerance for network latency. Excessive response time can have unpredictable consequences for devices, and even lives. Nowadays, how to reduce the impact of virtualization on predictability and low latency has become one of the most significant research topics.

## 3. Virtualization

In 1959, computer scientist Christopher Strachey introduced the concept of virtualization in the paper "Time sharing in large fast computers" [6]. In 1964, IBM creatively designed a new operating system (OS) called CP-40, which realized OS virtualization. It provided support for multiple OSs by encapsulating the computing resources of mainframes [7]. The real maturity of virtual computing systems is reflected in the IBM system/370 mainframe [8].

Overall, the development of virtualization has a history of 60 years. Virtualization technology is a technology that combines or partitions existing computer resources (CPU, memory, disk space, etc.) to make these resources appear as one or more operating environments, thereby providing better access methods than the original resource configuration.

This section describes some key similarities and differences between containers and virtual machines (VMs), as well as the respective usage scenarios for VMs and containers.

## 3.1. Virtual machine (VM)

VMs are typically measured in gigabytes. They usually contain their own operating system (OS), thus being able to perform several resource-intensive functions at once.

**3.1.1. Structure of VM.** Unlike containers, a VM runs a full OS (including its own kernel), as shown in the Figure 1 [9]:

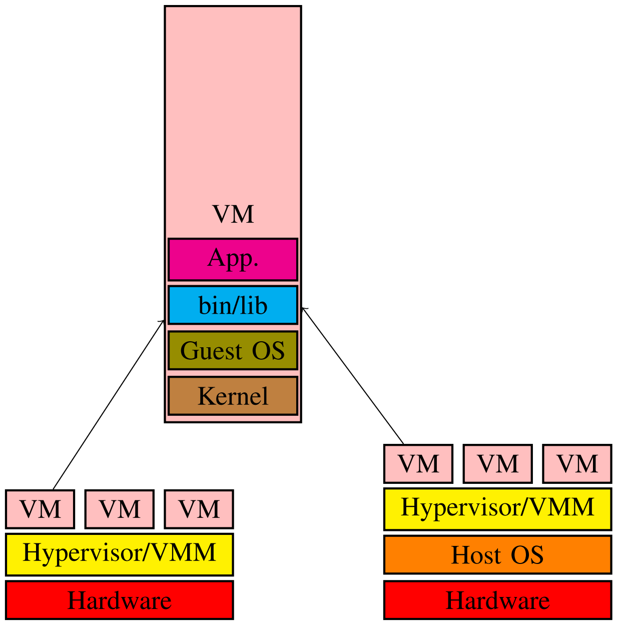

Figure 1: Structure of VM

From the bottom to the top:

1. **Infrastructure (Hardware)**: It is composed of CPU, Memory, Disk, NIC and other hardware. Infrastructure can be a laptop in the office, a server in a data center, or a cloud host in an enterprise.

2. **(Host Operating System)**: The Host OS, which is the original OS of the device, can be installed on the bare-metal environment (infrastructure). For VM, the Host OS is not necessary, whether there is a Host OS depends on the type of Hypervisor (Typ1 or Typ2).

3. **Hypervisor / Virtual Machine Monitor (VVM)**: It is an abstraction layer between the existing hardware and the other OSs that will be installed [10]. With Hypervisor, it is possible to operate multiple different Guest OSs independent of the hardware.
In 1973, Robert Goldberg distinguished two types of hypervisors in his PhD Thesis “Architectural Principles for Virtual Computer Systems” [11]. Type 1 is based on the hardware directly and do not require prior installation of an OS, e.g. HyperKit supports MacOS, and Hyper-V supports Windows. Type 2 is based on the Host OS above the infrastructure and uses the device driver of the OS to access the hardware, e.g. VirtualBox and VMWare Workstation [10].

4. **Kernel** and 5. **Guest Operating System**: As shown in the figure, suppose you need to run 3 isolated Apps, you need to use Hypervisor to start 3 Guest OSs, i.e. 3 VMs. These VMs are very large. Assuming that the size of each VM is 1.2GB, this means that they will take up 3.6GB of disk space. Even worse, at the same time they also consume a lot of CPU and memory.

6. **Dependencies**: Many dependencies need to be installed to each Guest OS, i.e. bin and lib files required by various Apps.

7. **Applications**: After installing dependencies, Apps can be operated on each Guest OS separately, i.e. each App is isolated and invisible from each other.

## 3.2. Container

Containers are typically measured in megabytes. They encapsulate no larger content than an App and all its required files. In addition, containers are often used to encapsulate individual functions that perform specific tasks, i.e. microservices [12].

**3.2.1. Structure of Container.** Containers are much simpler than VMs. Without the Guest OS, container can save disk space and other system resources, as shown in the Figure 2 [9]:

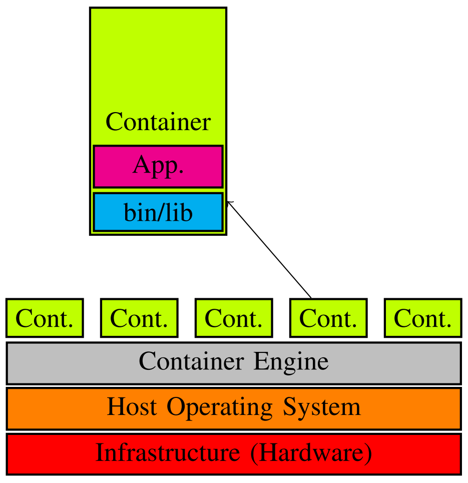

Figure 2: Structure of Container

From the bottom to the top:

1. **Infrastructure (Hardware)**: Introduction as in 3.1.1

2. **Host Operating System**: All major Linux distributions can operate containers, e.g. Docker. For macOS and Windows, it can also run Docker with a proxy.

3. **Container Engine**: It replaces the Hypervisor to communicate directly with the Host OS and allocate resources to respective containers. Container Engine can also isolate containers from the Host OS and isolate each container from each other [13].

4. **Dependencies**: A container encapsulates all dependencies of an App in an image, e.g. a Docker container is created based on a Docker image. When different containers share the kernel of the host OS, they cannot access it freely, but only its virtualized view, e.g. a container can

access a virtualized version of the file system and registry, but any changes only affect the container in which they reside and are discarded when the container is stopped [13].

5. **Application**: The source code of the App and its dependencies are encapsulated in the image, different Apps require different images. Different Apps are operated in different containers, they are isolated from each other.

## 3.3. VM vs. Container

For VM and container, the two most advanced virtualization methods, which is the preferred technology? This depends on the different application scenarios and requires.

**3.3.1. Application scenarios for VMs.** Compared to containers, VMs occupy much more resources and therefore VMs are able to run more operations. Furthermore, due to their greater isolation, VMs can run the Apps with business-critical data [4] and store traditional monolithic workloads [12], e.g. abstracting, separating, replicating and emulating entire servers, operating systems, desktops, databases and networks.

**3.3.2. Application scenarios for Containers.** Because of the lightweight and shared OS properties, containers are very convenient to move between multiple environments (high portability) [12]. They are more suitable for deploying cloud-native applications, i.e. a collection of many microservices, with the aim of providing consistent development management for Apps in a multi-cloud hybrid environment [4].

## 4. Optimization of measurements

This section describes methods for optimizing measurements in the process of quantifying latency. The purpose of optimization is to reduce the impact caused by human or physical hardware factors, i.e. delays that do not originate from the communication technology itself.

## 4.1. Accuracy vs. Precision

The quality of latency measurements can be evaluated in two dimensions: accuracy and precision [14]. For general measurements, accuracy will be considered as the magnitude of the difference between the timestamp of the measurement result and the actual occurrence of the event, i.e. the round trip time spent to transmit the data of the measurement result. Precision is defined as the statistical variability between measurements, i.e. the degree of variability and fluctuation of values between multiple measurement results.

**4.1.1. Hardware Timestamp.** Data reception on modern servers occurs asynchronously, i.e. the NIC receives the packet first, then it is past through random access memory (RAM). Finally, the reception is signaled to the CPU [14]. By default, the software timestamp will be generated after announcing the reception to the CPU. Due to the interrupts caused by system calls of the OS [15], the round-trip time increases, resulting in a delay in the software timestamp, which further generates a measurement latency and a significant decrease in the accuracy.

A feasible solution is to use hardware timestamps. After the packet is received by the NIC itself, the hardware device responds with the shortest clock to timestamp the packet accurately, since the hardware timestamp is not affected by OS interrupts [14].

**4.1.2. Measurement error tolerance.** According to a range of requirements of URLLC, the end-to-end user-level latency must be less than $1ms$, the control-level latency between critical infrastructure and computers must be less than $10ms$ and the communication reliability must be greater than 99.999% etc. [16]–[18]. Therefore, a slightly large error would cause a catastrophic impact on the latency measurement results. Low-precision measurement systems may seriously affect latency measurements due to their own introduced statistical errors.

The usage of measurement systems with the highest possible precision to obtain more precise measurement results is a basic prerequisite for scientific experiments.

## 5. Optimization of performance

This section introduces the available optimization methods to reduce the latency in the communication process. It also suggests potentially feasible latency optimizations that can be further explored in the future.

## 5.1. Available optimization

According to Gallenmüller et al. it was shown that measures such as kernel isolation, disabling virtual cores or disabling energy saving mechanisms and reducing the number of interrupts can be achieved by modifying parameters in the Linux system [14]. Using these mechanisms, the latency of packet-processing Apps can be reduced. This optimization method eventually achieves latency as low as $25\mu s$ in the worst case.

The whole process of the optimization method is complex. It requires the strict environmental configuration and changes various parameters. However, the central point of optimizing the latency is as follows.

**5.1.1. Disable virtual cores.** Disabling virtual cores can be achieved via the $nosmt$ kernel boot parameter [14]. It must be disabled to avoid the effects of unnecessary resource contention between different virtual cores.

**5.1.2. Enable core isolation.** The isolation is enforced by the parameter $isolcpu$ which prevents the Linux scheduler from scheduling processes to a specific kernel [14]. With this configuration, neither the Host OS nor the VMs' OS can schedule processes to the kernel of the executing App. Therefore, core isolation is achieved which creates the perfect environment for the uninterrupted execution of packet-processing Apps.

**5.1.3. Reduce the number of CPU OS interrupts.** However, just using core isolation remains ineffective in reducing latency. In the research [19] it was found that not handling OS interrupts that occur on isolated cores still leads to latency peaks of up to about $20\mu s$. Therefore, it is necessary to disable scheduling interrupts when running a single App on a specific core by the parameter $nohz\_full$ (No timer ticks). In addition, two option parameters $rcu\_nocbs$ (No RCU callbacks) and $rcu\_nocb\_poll$ (No RCU callbacks threads wackup) are needed to transfer the intra-core read-copy-update (RCU) processing to different cores, thus avoiding interrupts [14].

Furthermore, the NIC can interrupt processing to signal the reception of new packets. By setting the parameter $irqaffinity$ to 0, data can be processed on the specific core. The Data Plane Development Kit (DPDK) can also be used to reduce the impact of the Linux kernel on network Apps in order to achieve high reliability and low latency on hardware and virtualized systems [19].

**5.1.4. Disable energy saving mechanism.** To keep the CPU always in the most active state, the options $idle$ (Poll mode when core idle) and $intel\_idle.max\_cstate$ (Limit CPU to c-state) can be used. In addition, $intel\_pstate$ needs to be disabled to prevent the driver from switching the CPU to a power saving state [14]. Disabling the energy saving mechanism improves latency above 99.99% by about $10\mu s$ [20].

## 5.2. Potential optimization

The latency optimization methods mentioned above are in the face of VM-based virtualization technologies. However, as introduced, containers require fewer resources than VMs and have the properties of lightweight and shared OSs, which provides the chance to improve the efficiency of communication between multiple environments dramatically.

**5.2.1. Container-based virtualization optimization.** Moreover, different services can be implemented by slicing the network into different independent logical networks (network slicing) [19]. An efficient way to realize network slicing is to share the usage of network resources among clients. The small size and high portability of containers meet the demand for deploying numerous microservices.

As Ann Mary Joy found in her study [4], in the same configuration environment Docker containers are able to process more requests in the same amount of time. It is more than 5 times more than VMs, because containers take less time to process a single request. In addition, containers take far less time to scale and process service requests than VMs do. Containers have great scalability, about 22 times of VMs. Even in extreme cases, such as when a container crashes during operation, which affects the performance of other containers, their performance is still much better than VMs [21].

Intuitively, it can be predicted that the usage of containers instead of VMs in the virtualization process will yield better performance.

**5.2.2. Container- & MV-based Hybrid virtualization optimization.** The other potentially viable optimization option is a hybrid virtualization approach that uses a VM as the lightweight host OS, with the upper layer interacting further with containers. By leveraging the benefits of each, the performance of communication can be predictably further improved. Although experimental results from one survey have shown that the performance of VMs mixed with containers is unimaginably worse [22]. However, in fact, many container deployments use VMs as host OS rather than running directly on hardware, especially when running containers in the cloud [13].

## 6. Conclusion

In conclusion, the core meaning of virtualization technology is to enable computing environments to run multiple independent systems simultaneously [23]. Both containers and virtual machines (VMs) are encapsulated computing environments that combine various IT components and are independent of the rest of the system. The main differences between the two are the magnitude, scalability and portability [4]. URLLC has very demanding requirements for low latency. Since the units of measurement values are inherently at the millisecond or microsecond level [16]–[18], very small measurement values will further reduce the tolerance interval for errors. Therefore, improving accuracy and precision is essential for reliable latency measurements. This also further increases the difficulty of combining virtualization and low latency. From the latency measurement results, the performance of the low-latency optimization on both bare metal and VMs in the lab's operating environment is satisfactory. By disabling virtual cores and energy saving mechanism, as well as reducing the number of CPU interrupts, the latency peaks can be significantly reduced even in extreme cases.

Nowadays, container-based virtualization technology is maturing. In contrast to VMs, containers are lightweight, easier to deploy, require fewer resources and are more efficient. This offers the possibility to reduce the latency of virtualization continuously. However, as an emerging technology, containers are poorly isolated. Only the CPU can be well isolated, but the memory, disk and network cannot be ideally isolated [24]. This also presents hidden risks to the reliability of communication. Currently, there are few solid arguments in favor of "Container-based virtualization leading to more reliable and lower latency communications". I plan to review and research further in the future to determine the feasibility of this implementation.

## References

[1] "Minimum requirements related to technical performance for IMT-2020 radio interface(s)," *ITU*, 2017, [Online; accessed 17-Dec-2022].

[2] D. Patel and J. Sachs, "5G E2E Technology to Support Verticals URLLC Requirements," *NGMN Board, Verticals URLLC Requirements*, 2020, [Online; accessed 1-Dec-2022].

[3] W. Gerlach, W. Tang, K. Keegan, and T. Harrison, "Skyport - container-based execution environment management for multi-cloud scientific workflows," pp. 25–32, 2014.

[4] A. M. Joy, "Performance Comparison between Linux Containers and Virtual Machines," *International Conference on Advances in Computer Engineering and Applications, 2015*, pp. 342–346, 2015.

[5] J. Ott and C. Eckert, "Slide, Prozess- und Prozessorverwaltung, Einfuehrung," 2022.

[6] C. Strachey, "Time Sharing in Large Fast Computers," *Proceedings of the International Conference on Information Processing, UNESCO*, 1959.

[7] R. Creasy, “The Origin of the VM/370 Time-Sharing System,” *IBM Journal of Researchand Development*, vol. 25, no. 5, pp. 483–490, 1981.

[8] L. H. Seawright and R. A. MacKinnon, “VM/370—A Study of Multiplicity and Usefulness,” *in IBM Systems Journal*, vol. 18, no. 1, pp. 4–17, 1979.

[9] “Windows and Containers,” https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/, 2022, [Online; accessed 1-Dec-2022].

[10] J. Ott and C. Eckert, “Slide, Prozess- und Prozessorverwaltung, Virtualisierung,” 2022.

[11] R. P. Goldberg, “Architectural Principles for Virtual Computer Systems,” 1973.

[12] “Introduction to Virtualization, Brief History of Virtualization,” https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html, 2012, [Online; accessed 1-Dec-2022].

[13] “Containers vs. Virtual Machines,” https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm, 2021, [Online; accessed 1-Dec-2022].

[14] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, “Ducked Tails: Trimming the Tail Latency of(f) Packet Processing Systems,” *17th International Conference on Network and Service Management (CNSM), 2021*, pp. 537–543, 2021.

[15] A. S. Tanenbaum, *Modern Operating Systems*, fourth edition, global edition ed. Boston, Pearson, 2015.

[16] G. Brown, “Ultra-Reliable Low-Latency 5G for Industrial Automation,” *A Heavy Reading white paper produced for Qualcomm Inc.*, 2019, [Online; accessed 1-Dec-2022].

[17] Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh, “5G URLLC: Design Challenges and System Concepts,” *Institute of Electrical and Electronics Engineers*, pp. 1–6, 2018.

[18] R. Ali, Y. B. Zikria, A. K. Bashir, S. Garg, and H. S. Kim, “URLLC for 5G and Beyond: Requirements, Enabling Incumbent Technologies and Network Intelligence,” *Institute of Electrical and Electronics Engineers*, pp. 67 064–67 095, 2021.

[19] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, “5G QoS: Impact of Security Functions on Latency,” *IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2020.

[20] M. Primorac, E. Bugnion, and K. Argyraki, “How to Measure the Killer Microsecond,” *Comput. Commun. Rev.*, vol. 47, no. 5, pp. 61–66, 2017.

[21] M. G. Xavier, I. C. D. Oliveira, and F. D. Rossi, “A performance isolation analysis of disk-intensive workloads on container-based clouds,” *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015*, pp. 253–260, 2015.

[22] B. Ruan, H. Huang, S. Wu, and H. Jin, “A Performance Study of Containers in Cloud Environment,” *Wang, G., Han, Y., Martínez Pérez, G. (eds) Advances in Services Computing. APSCC 2016.*, vol. 10065, 2016.

[23] “Introduction to Virtualization, Brief History of Virtualization,” *oracle*, 2012, [Online; accessed 1-Dec-2022].

[24] M. G. Xavier, M. V. Neves, and F. D. Rossi, “Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments,” *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2013*, pp. 233–240, 2013.