applied sciences

MDPI

Article

Docker Performance Evaluation across Operating Systems

Maciej Sobieraj * and Daniel Kotyński

Institute of Communication and Computer Networks, Faculty of Computing and Telecommunications, Poznan University of Technology, ul. Polanka 3, 60-965 Poznań, Poland
* Correspondence: maciej.sobieraj@put.poznan.pl

Abstract: Docker has gained significant popularity in recent years. With the introduction of Docker Desktop for Windows and macOS, there is a need to determine the impact of the operating system on the performance of the Docker platform. This paper aims to investigate the performance of Docker containers based on the operating system. One of the fundamental goals of this study is to conduct a comprehensive analysis of the Docker architecture. This technology utilizes Linux kernel virtualization mechanisms such as namespaces and cgroups. Upon analyzing the distribution of Docker Desktop for Windows and Docker Desktop for macOS, it was discovered that running the Docker environment on these requires a lightweight virtual machine that emulates the Linux system. This information suggests that the additional virtualization layer may hinder the performance of non-Linux operating systems hosting Docker containers. The paper presents a performance test of the Docker runtime on Linux, Microsoft Windows, and macOS. The test evaluated specific aspects of operating system performance on a MacBook computer with an ×86/64 processor architecture. The experiment carried out examined the performance in terms of CPU speed, I/O speed, and network throughput. This test measured the efficiency of software that utilizes various system resources.

Keywords: Docker; operating system; performance tests; virtualization

Citation: Sobieraj, M.; Kotyński, D. Docker Performance Evaluation across Operating Systems. *Appl. Sci.* **2024**, *14*, 6672. https://doi.org/10.3390/app14156672

Received: 14 June 2024
Revised: 23 July 2024
Accepted: 30 July 2024
Published: 31 Junly 2024

1. Introduction

The increasing demand for scalable and efficient software has made virtualization a desirable technology in the contemporary IT world. Docker—one of the popular virtualization technologies—is used by more than 18 million developers worldwide and ranks as one of the most beloved tools [1–3].

One of the most crucial aspects that affect the performance of any virtualization runtime is the underlying operating system of the host. Among all supported operating systems, Linux-based are often considered to be the best in terms of user experience and performance [4,5]. As variations in operating system architectures and runtime implementation may cause differences in the speed of Dockerized applications, the choice of operating system could be essential.

Performance evaluation of deploying dockers was a subject of many works [6–11]. In [6] studies about the evaluation of Apache and Nginx web server performances on the Docker platform were presented. Two separate Docker containers were used in the experiments. Six web server performance criteria were evaluated: requests per second, failed requests, availability, failed transactions, response time, and transaction rate. The work [7] proposes the system which calculates the startup time for Docker containers running the Apache web server in bare metal and virtual machine (KVM). The analysis helps to decide the environment to work with the Docker application to obtain maximum performance. In [8] authors conduct a comparative study of the performance evaluation of virtual machines and containers. A methodology to evaluate the performance of Docker containers and virtual machines was also proposed. In addition, a real-world case study was presented to illustrate the applicability of the proposed approach. Another evaluation, presented

in [9], concerns the impact of docker container on the performance of deep learning applications. Authors benchmark the performance of system components (IO, CPU and GPU) in a docker container and the host system. The next problem that was addressed in [10] was the container storage system. Authors analyze terabytes of uncompressed Docker Hub images, characterize them using multiple metrics, and evaluate the potential of file-level deduplication. The analysis helped to make conscious decisions when designing storage for containers in general and Docker registries in particular. The main goal of the paper [11] was the performance comparison and potential problems of container-based virtualization, with Docker and Podman as typical representatives. The authors evaluated the impact of one and more container-based virtual machines on file system performance. So we can see that the topic of Dockers is very popular. There are still many docker related issues that need to be analyzed.

The aim of this paper is to create and use a performance test of the Docker runtime on Linux, Microsoft Windows, and macOS. This experiment uses MacBook Pro 13 (2019) with an Intel CPU as the hardware platform for benchmarks.

There is no official Docker guideline for recommended operating systems. The fact that Docker implementation depends on Linux kernel features to provide virtualization, and before 2016 it was not available on other platforms may suggest that Docker will achieve the greatest performance on Linux. The result of this paper may be utilized to prepare recommendations for choosing an appropriate operating system when working on Docker containers. This recommendation could benefit Software Engineers, System Administrators, and DevOps professionals, among others, who work with Docker containers.

The main objective of this paper is to carefully evaluate the performance overhead of different operating systems when hosting Docker containers. This study's subgoals are:

- Prepare the testing environment that will allow fair and accurate performance measurements.
- Design various benchmarks, the results will not only show performance differences but also point to potential bottleneck points.
- Identifying trade-offs of each solution.
- Recommend the best environment based on the test results.

This paper is organized in the following structure. Section 2 is a theoretical introduction to virtualization, the principles of containers, and the mechanisms used by the Docker implementation. Section 3 outlines the adopted testing methodology and describes the platform used for the experiments. Section 4 reports the obtained results and provides a preliminary analysis. Section 5 is a summary with a conclusion preceded by an analysis of all observations and a description of ideas for future research.

2. Docker Architectures

Docker is available for all three platforms that are being tested in this paper, but it's necessary to distinguish Docker from Docker for Windows and Docker for Mac, as they differ in implementation. Docker is only available on Linux, as it uses Linux kernel virtualization features. Docker for Windows and Docker for Mac both come with some additional virtualization techniques that might create performance overhead.

2.1. Docker on Linux

To describe how Docker works on Linux, we would say that it uses low-level virtualization features to provide high-level virtualization. Those features are Control Groups (cgroups) and namespaces [12,13].

2.1.1. PID Namespace

The namespace mechanism was introduced in the Linux kernel version 2.4.19. The first implementation included Process ID namespaces. PID namespace creates the illusion that processes in a given namespace are the only ones in the system. Each process namespace comes with its own PID set.

As seen in Figure 1, a PID = 6 process called an unshare [14] command with the PID flag resulted in the creation of a new PID namespace. Two processes with PID of 9 and 10 from the parent namespace perspective are seen as PID 2 and 3 processes (green color) from the child (PID 8) process perspective. PID namespaces can be nested—processes mentioned earlier could also create new namespace; this feature can be used in "A container inside a container" kind scenarios.

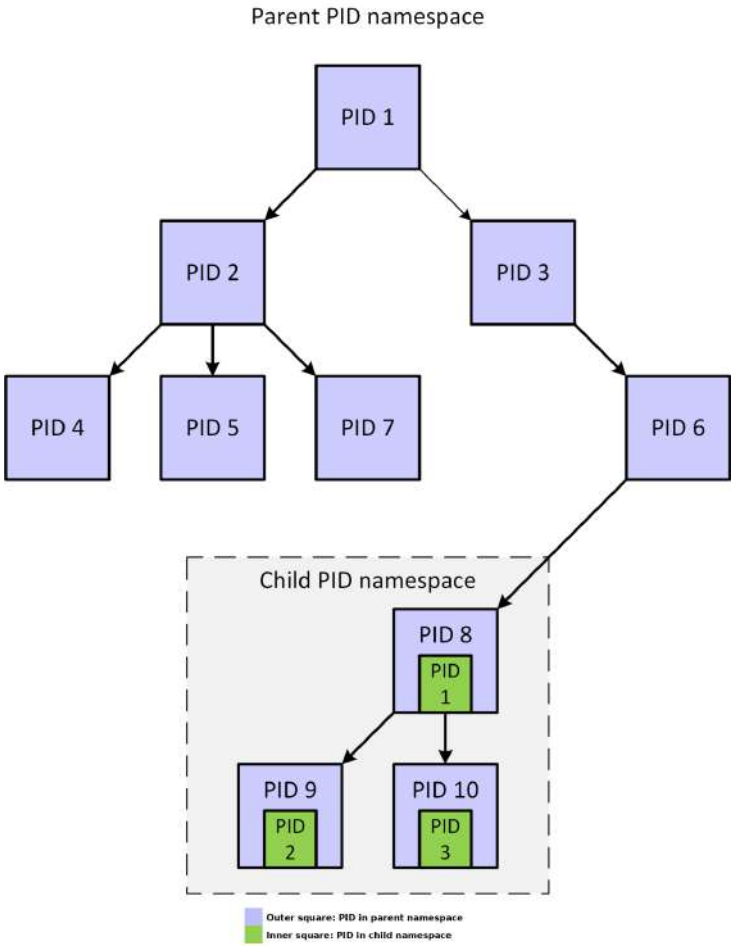

Figure 1. Linux PID namespace visualization.

When studying PID namespace mechanisms, it is worth mentioning the special behavior that is helpful when implementing container environments. The process with PID = 1—which is called the init process is the parent of all processes running on a Linux machine [15]. For this reason, the init process does not terminate after receiving SIGTERM or SIGKILL signals in most cases to prevent unwanted system termination [16]. This feature is utilized in Docker containers, where in the initial process in the designated namespace, it cannot be easily terminated [17].

These properties allow Docker containers to isolate container processes from the host, even making Docker container instances not aware that they are isolated by Docker. Each process spawned in a container is in fact spawned at the host machine but in a separate container namespace. A shell running in a container can access only processes in this container, while processes spawned by a Docker container can be seen on the host using ps [18] command. We can observe how the process namespace creates a new PID set, since the same process from the host perspective has a different PID from that one in the container perspective. Later Linux kernel versions brought other namespaces, described in the following.

2.1.2. Mount Namespaces

Docker uses mount namespaces to separate the container file system [19]. Each mount namespace specifies which mount points can be seen by the container. This namespace provides isolation between the host and the container and between multiple containers. The original implementation of mount namespaces has shown that the isolation it provided was sometimes too excessive. This issue was fixed by providing mount propagation types [20].

2.1.3. Network Namespaces

Another namespace used by virtualization tools is the network namespace. This feature allows each container to have its own routing table, ARP table, and network device with IP address.

Docker networks utilize Linux network namespaces in such a manner that constructing multicontainer applications—especially using docker compose—is easy and convenient. Docker network provides such features as isolation, where users can define inter-container connections. Containers can communicate in the same network using their names thanks to internal DNS resolution. Ports of docker containers can be exposed for external access, and mapped to avoid port collisions.

2.1.4. User Namespace

An important feature of Docker containers in terms of security and isolation is the ability to map the UID in a container to a higher value of the user ID on the host system. Using a user namespace can prevent container breakout, even when a user in a container has root privileges. A high UID that is mapped to the host system prevents harmful actions as a user with such a UID does not have enough privileges. This feature also helps to distinguish container processes from host processes.

2.1.5. IPC Namespace

Responsible for isolating interprocess communication, structures such as messages, queues, and semaphores are separated between containers and host.

2.1.6. Control Groups

In addition to isolation, which is provided mostly through the use of namespaces, a decent virtualization tool needs to use system resources efficiently. Control Groups allow the regulation of CPU, memory, disk and network I/O division between multiple containers and a host machine [21]. This is used to prevent over-consumption from a single container and avoid resource contention. It is possible to achieve predictable performance by limiting resource usage. It is important to know that by default containers can access resources as long as they are available on the host.

Docker distinguishes two types of limits: soft limit and hard limit. The hard limit specifies the amount of a given resource that a container cannot exceed, while the soft limit defines a limit that works as a hard limit but only, when Docker detects contention or low memory on the host machine. A soft limit must have a lower value, and the container can exceed those limits in opposition to the hard limits [22].

In order to appreciate cgroups usage in the Docker system, it is important to understand the consequences of OOME—Out Of Memory. When the physical and swap memory of the Linux system is exhausted, the Linux kernel launches the OOM process killer, which is responsible for terminating processes to prevent a system crash [23]. In Docker containers, this idea is extended to individual containers. When an instance of a container runs out of memory, process termination is executed in a separate namespace and avoids interrupting apps in other container and host systems [24].

2.2. Docker for Windows

This Docker distribution is used to run Linux images on selected Windows versions; currently, Windows 10 and Windows 11 are supported. One of the Docker for Windows dependencies is WSL2—Windows Subsystem for Linux [25].

WSL is a lightweight virtual machine that uses Hyper-V Hypervisor. Docker is often compared as an alternative to virtual machines. It uses one on Windows machines. It is worth mentioning that only one instance of virtual machine is being used, although many independent containers might be running on a Windows host. Despite using virtual machines, WSL2 is considered a lightweight virtualization, with barely noticeable performance loss (94 percent when comparing WSL2 with native Linux [26]). Theoretically, this architecture should not outclass native Linux Docker installation, although it still should provide greater performance when compared to traditional VMs, especially when running multiple applications.

Another key component that is part of Docker for Windows is VPNkit [27]. It is needed to provide connectivity between external networks and containers respecting the host network configuration, such as VPN and DNS. It works by translating Ethernet traffic from a container to the host's socket API calls.

2.3. Docker for Mac

Docker Desktop for Mac has many similarities to the Windows distribution. Docker Desktop for Mac utilizes the Moby framework, which provides Linux Kernel features, on which Docker is built [28]. It also uses VPNKit to handle connectivity. In contrast to Docker Desktop for Windows, it runs under HyperKit instead of the Hyper-V hypervisor. Docker supports Docker Desktop on the latest versions of macOS, specifically the current release and the two preceding ones. When a new major version of macOS is released, Docker drops support for the oldest version and begins supporting the newest version, along with the two prior versions.

3. Methodology

Achieving reliable, relevant, and replicable results requires designing a robust methodology. Selecting proper procedures should ensure that gathered data will be consistent and comprehensive. This section describes selecting an adequate measurement method.

3.1. Test Platform

In order to obtain meaningful results, it is important to choose the appropriate test platform, the operating systems on which Docker will be run, and how it will be configured.

3.1.1. Choosing Testing Platform

To provide accurate and meaningful measurements, selecting an appropriate test platform is significant for the value of the results. To guarantee identical computing power, the test platform's hardware should support all operating systems being tested. However, the macOS EULA prohibits the installation of the OS on non-Apple hardware [29], requiring the use of an Apple computer for testing purposes. As $\times 86/64$ is a much more commonplace CPU architecture [30] and running ARM images on $\times 86$ requires a translator, leading to performance overhang. The test platform shall utilize an $\times 86/64$ architecture-based operating system along with container images. Apple MacBook Pro with Intel chip was used in tests that met all requirements specified in this section (Table 1).

Table 1. Hardware specification of test platform.

CPU	RAM	Storage
Intel i5-8257u @ 1.40 GHz	16 GB LPDDR3 2133 Mhz	256 GB NVME SSD

3.1.2. Used Operating Systems

The operating systems used in the research are shown in Table 2. The macOS Ventura, Windows 10 and Ubuntu were used in experiments as the most popular operating systems and widely used in professional applications. MacOS Ventura was used as is was the latest release at the moment of conducting the study (which was the second half of 2023). MacBook Pro 13—the machine that serves as the testing platform has a built-in T2 chip [31]. One of its functions is to protect the boot process from injecting a malicious operating system into the computer. As this feature improves security, it hinders the installation of untrusted systems. From the point of view of T2, Ubuntu is considered unsafe, as Apple does not support installing the GNU/Linux operating system on its machine. This barrier could be overcome with the help of the T2-Ubuntu [32] distribution, designed to be installed on Apple computers with T2-chipped. The Ubuntu version used in this study was based at latest LTS release at the time of conducting the study, which is 22.04. The authorized method for installing Windows on Apple hardware involves utilizing Boot Camp Assistant 6.1.19 software, which is exclusively compatible with Windows 10 [33].

Table 2. Used operating systems.

System Name	Version
Windows 10	22H2
Ubuntu	22.04-6.4.8-t2-jammy
macOS	Ventura 13.5.1

3.1.3. Docker Settings

In order to create environments that allow fair performance comparison, Docker must access uniform hardware resources. While Docker on native Linux will utilize all system resources when necessary, and Docker Desktop distributions provide a way to limit the resources provided for the docker engine, Docker Desktop on Mac and Windows should utilize all its resources. It is important to know that by default only part of the resources were available for Docker Desktop implementation. It is possible to alter this setting using the Docker Desktop user interface on macOS and by modifying the .wslconfig [34] file on Windows. This test platform utilizes 8 logical CPUs, 16 GB of RAM, and 1 GB of swap memory. The version of the Docker Engine that is used on the testing platform is 4.20.

3.2. Test Methodology

A proper test methodology should be applied to measure the performance of the system and the virtualization overhead. This test consists of fine-grained benchmarks that aim to measure only a given part of the operating system and more complex tests that put stress on multiple levels, recreating performance differences in real-case scenarios. To achieve accurate measurements, the test is run on a freshly installed operating system, without any unnecessary background services that are not required for the benchmark to operate.

3.3. Calculating Pi Number

π number calculation is a popular way to directly test computing performance. This benchmark utilizes the Leibniz formula for π (Equation (1)) implemented in C language. The tested setup used a Linux Alpine base image with the GCC compiler. An executable binary was created using the -03 flag for greater performance. Each test ends after 2^{32} iterations.

$$\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots\right) \quad (1)$$

3.4. Sysbench

Sysbench is an open source tool that is capable of measuring various components of a system. This software provides standardized and reproducible test scenarios that could

provide meaningful outputs about systems performance [35]. The versatility of this tool could benefit from measuring many aspects of tested environments. Although this tool provides its own scripting language to design custom benchmarks, this test will make use of built-in scripts [36].

CPU test will target the amount of operation in a given time that the CPU can handle with all its logical cores. This test utilizes another popular computation technique that is widely utilized when determining CPU speed—prime number calculation.

The input-output experiment is going to target hard drive operation performance in different modes. Modes that will be tested are:

- sequential read,
- sequential write,
- random read,
- random write,
- random combined write and read.

Each single test is preceded by a preparation process, which creates a certain number of files. To provide meaningful outputs for each test, there is a cleanup method that deletes test files to avoid any caching.

3.5. Iperf3 Tool

Iperf3 is a network performance measurement tool that creates controlled data flow between two instances in a network. The primary purpose of this program is to gather information on network connection performance such as throughput, jitter, latency, and efficiency. This benchmark utility is designed to operate in a client-server architecture. This benchmark's primary goal is to determine Docker network performance between containers and a host. The iperf3 scenario will cover three cases:

1. inter-container TCP,
2. inter-container UDP,
3. client on host—server on container TCP.

To provide accurate and replicable test results, the time of program measurements has been extended to 60 s. The metrics that will be measured are throughput and packet loss. The bridge network driver was used in all three test cases. To provide connectivity between the host and the container, port 5201 has to be exposed by a server in the container.

3.6. Complex Benchmarks

The following class of tests consists of measuring the performance of test cases designed to replicate real-use scenarios of docker container use. These tests estimate the efficiency of the whole system without focusing on its components.

7zip is widely used as a popular archive tool. This experiment shall utilize the built-in benchmark function of 7zip, which uses different compression and decompression methods to determine the speed of the system.

PostgreSQL relational database is one of the most widely used docker images. The test will use pgbench—a tool designed to measure the performance of the PostgreSQL.

3.7. DoS

Higher-performance systems are generally more capable of resisting Denial-of-Service attack (DoS attack). This test will take advantage of this phenomenon. Measuring availability provides an indirect way to estimate performance. It is crucial to design such an attack in order to only lower system availability instead of complete denial of service. Making all three test instances completely unavailable will not provide useful information about performance differences. To perform a DoS attack, a separate machine is needed to avoid resource connection between attacker and victim processes. The machines were connected using a wireless access point.

3.8. Apache Slowhttp Attack

Apache is an open-source web server that will provide a static HTML website containing 50 paragraphs of Lorem Ipsum placeholder test. The objective of this benchmark is to perform a DoS attack using Slowloris, targeting an HTTP port exposed by the container on the host. The attacking machine is going to slow down container response by opening a massive amount of HTTP connection barely throughout, to keep the session open and make the server busy. This attack configuration consists of launching 800 sockets. Each HTTP request was delayed with an interval of 3 s to faithfully reproduce real user behavior.

4. Visualization and Analysis

The subsequent sections will present a detailed review of the visualized data, drawing attention to notable patterns and anomalies. The analysis shall focus on cases with significant differences between operating systems, although achieving a similar result for a given benchmark type is also valuable information. The Matplotlib charts were used to present the data from experiments.

4.1. Pi Calculation in C

As seen in Figure 2 using the Leibnitz method has pointed to the Windows system as the worst system for one-threaded CPU-based calculations. Linux with macOS with nearly identical scores outran Windows by more than one second on average. Lower execution time means greater CPU speed.

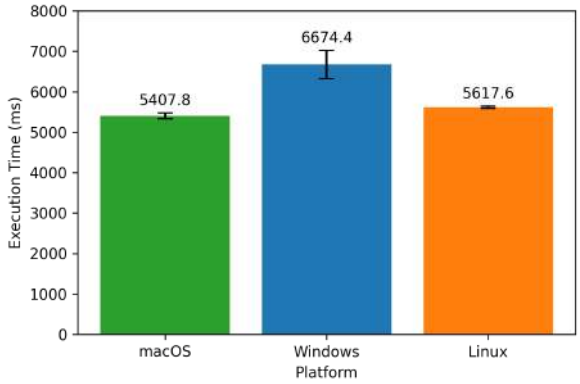

Figure 2. Leibnitz method π calculation in C.

4.2. Sysbench

CPU Sysbench test visualized in Figure 3 showed a visible performance advantage of the Linux operating system. Windows has a light—yet noticeable—performance overhead when comparing it to the environment with a lack of a virtual machine. macOS has the worst CPU speed, but the difference is not critical in terms of its usefulness.

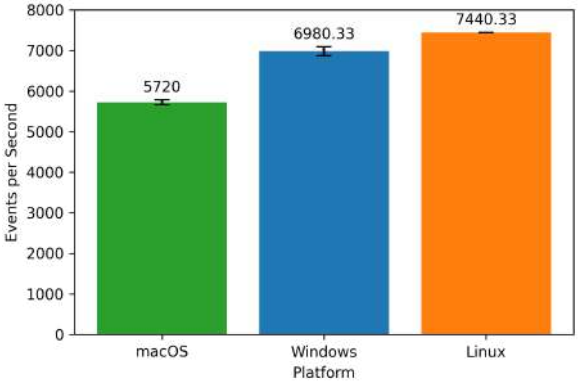

Figure 3. Sysbench CPU test.

The results of Sysbench i/o operations in Figure 4 showed comparable disk performance, with Linux as the system that handles reading memory slightly better than its competitors. When evaluating the writing speed shown in Figure 5 Ubuntu shows its fragility, especially in random writing. macOS outperforms Ubuntu by a substantial margin, exceeding its performance by a factor exceeding 40. The combined random read/write displayed in Figure 6 confirms this flaw in the Linux platform. Windows operating system has a related weakness in random write speed, with significant performance loss compared to the native system shipped with the testing platform. This difference can be explained by the poor SSD driver support of non-native systems.

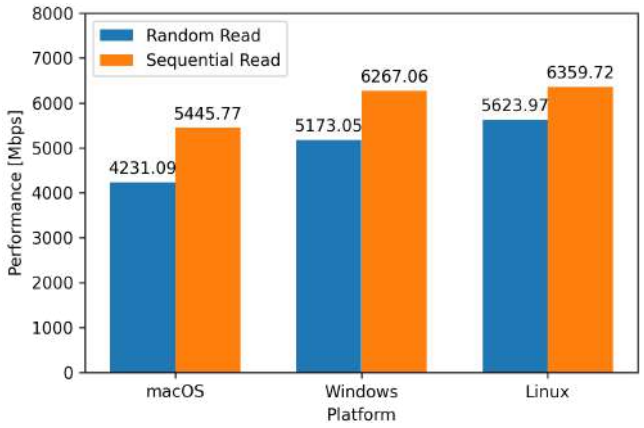

Figure 4. Sysbench I/O test of sequential/random read.

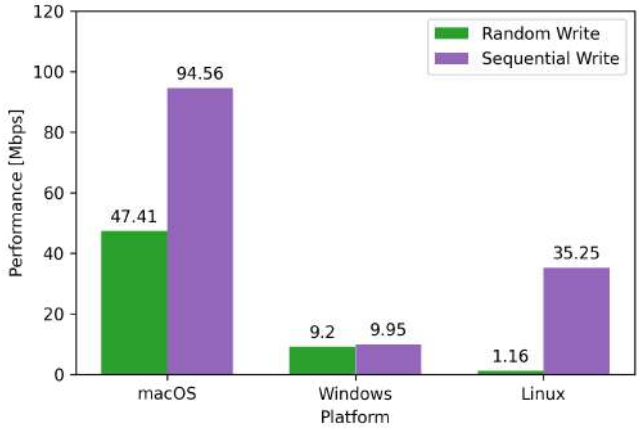

Figure 5. Sysbench I/O test of sequential/random write.

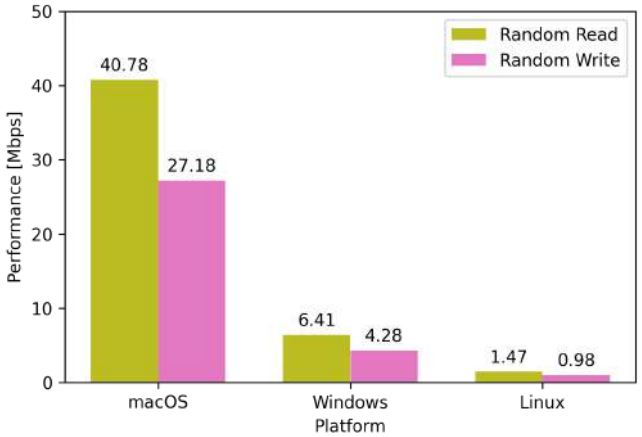

Figure 6. Sysbench I/O test of combined random write/read.

4.3. Iperf3

In the case of intercontainer communication, the performance of Windows is excellent, making it a great option when running network-oriented applications. It is outstanding in the throughput of TCP (Figure 7) and UDP (Figure 8), with loss of UDP packets 0% (Figure 9). As UDP packet loss is present on Linux and macOS, it is not considered significant. The performance of the Linux network between the host and the container outperforms other platforms, as seen in Figure 10. This characteristic proves that Linux containers are a great tool for running services for host-based applications.

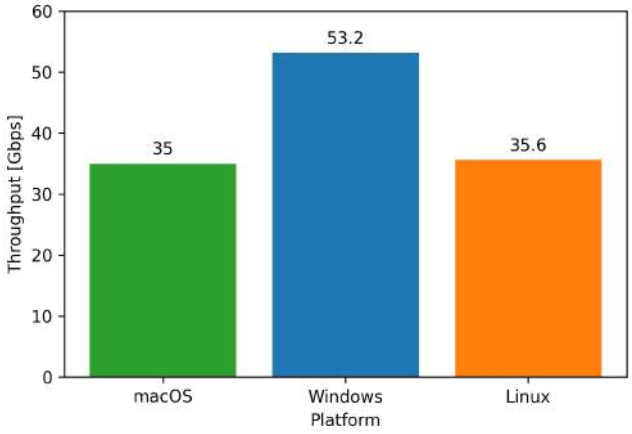

Figure 7. Throughput test between two containers using TCP protocol.

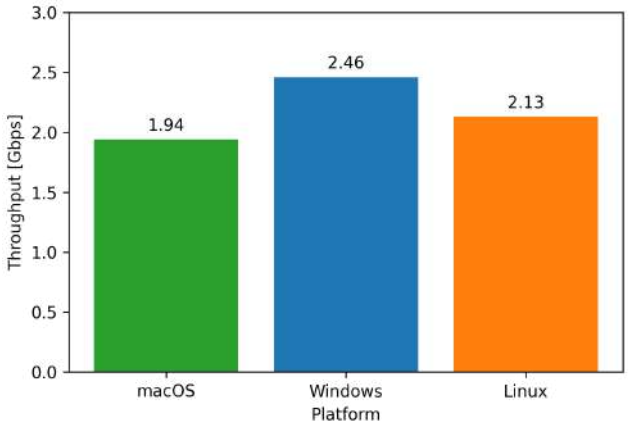

Figure 8. Throughput test between two containers using UDP protocol.

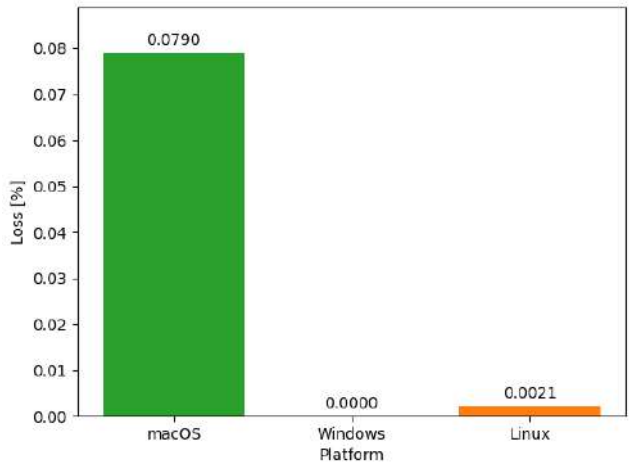

Figure 9. Packet loss test between two containers using UDP protocol.

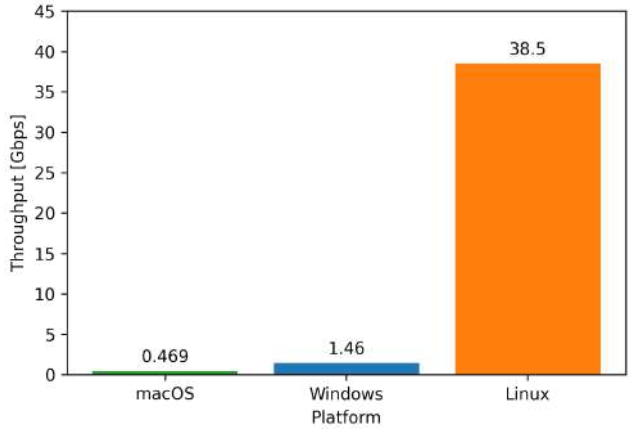

Figure 10. Throughput test between host system and container using TCP protocol.

4.4. 7zip

The archive tool test does not determine a winner in this category. With three different benchmark setups (Figure 11): single core, 4 cores (physical core count) and 8 cores (logical core count) only the last test pointed to Linux as a system that provides noticeable performance gain.

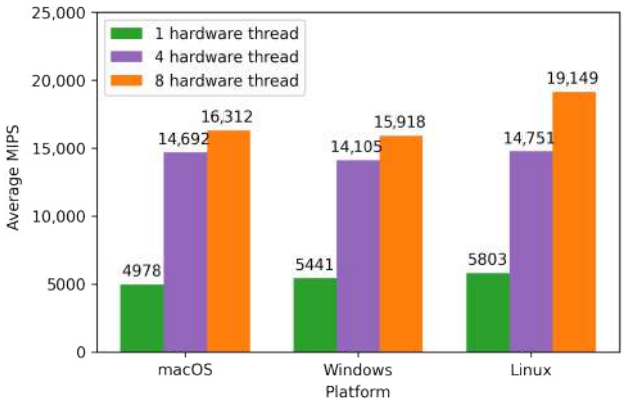

Figure 11. 7zip built-in benchmark score—using 1/4/8 hardware threads.

4.5. Pgbench

The pgbench result revealed a substantial disparity in performance across the platforms examined. PostgreSQL relational database running inside a Docker container on a macOS system has approximately 20,000% performance advantage in terms of transactions per second. This trend shown in Figure 12 has a strong similarity to the one observed while testing random writing in Figure 5 and combined random write/read in Figure 6. Those results might imply that a huge performance issue in PostgreSQL is related to a random write speed bottleneck on Windows and Linux machines.

4.6. Apache

To correctly interpret the outcome of this test, it is important to define the effectiveness of a partial Denial-of-Service attack. Each spike in Figure 13 represents an HTTP request that took a significant amount of time to finish. Each visible spike or point above a given threshold can be interpreted as an effect of partial denial of service. By measuring the ratio of requests that are executed in a long time to all requests, it is possible to determine which attack was the most effective. The outcome shows that Apache running inside Docker on the Linux host was the most affected target of the slow Loris attack. This benchmark was designed to show computing capabilities difference, but it does not correlate with any CPU or I/O based benchmarks. As network-based Denial-of-Service attack benefits from

the high throughput of the victim network, this test can prove great container-to-network connectivity on the Linux operating system.

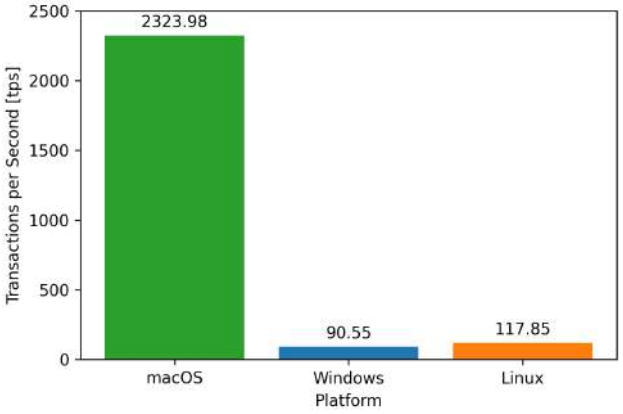

Figure 12. pgbench score for a database with the scale factor of 5.

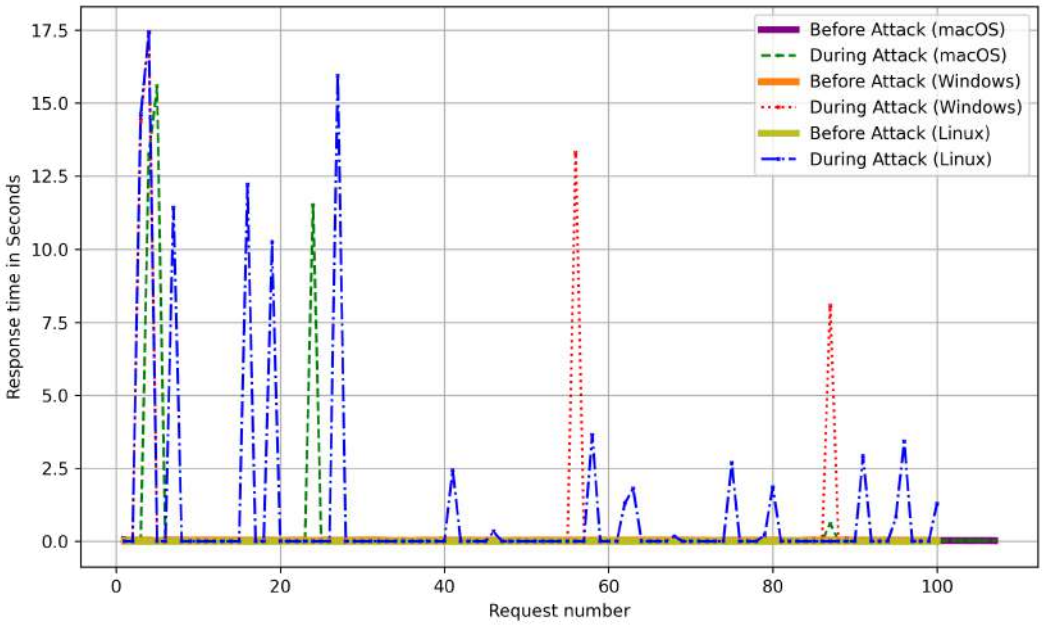

Figure 13. Response time of Apache server before and during a Slowlorris attack.

5. Conclusions and Future Work

By examining all results of the test procedure, it is clear that the operating system has a great influence on the behavior of the Docker container in terms of performance. The examined configurations appear to influence distinct aspects of container performance, such as CPU speed, memory operations, and networking throughput.

Across all operating systems tested, some benefited in a given category, while encountering serious performance loss in other categories. A structure can be as quick as its slowest component; therefore, it is extremely important to balance performance on each individual aspect of the system.

5.1. macOS Verdict

From the data gathered on our configuration, macOS is considered the most versatile system for operating with Docker containers. It does not suffer from any significant bottlenecks besides the host-to-container network speed visualized in Figure 10. As value of this throughput—469 Mbps—cannot be considered low, with a high probability it would match most requirements on a nonproduction setup, making Mac OS a solid operating

system for a container-driven workflow. It must be pointed out that this system is not a silver bullet, and there are cases where other systems will benefit more.

5.2. Linux Verdict

Despite the preference for macOS as the most versatile host system for Docker containers, it might not always be the optimal choice. In instances where a program rarely utilizes write-to-disk functionality, Linux may prove as a more efficient solution. In-memory databases, data streaming platforms, database cache, static websites, FTP read-only servers are among the systems that infrequently require write operation and can benefit from being run in a Docker container on a Linux host.

5.3. Windows Verdict

Windows system does not benefit from any particular test case except for intercontainer network throughput. This unique feature can be utilized in network-based applications, such as distributed systems. Similar to Linux, this setup's major drawback is the write speed; therefore, containers with high disk usage are not recommended for this configuration. Containers with Windows as host system could be used in applications recommended for the Linux system.

5.4. Critical Examination of Testing Methods

Due to the methodology of this test, there is a possibility that some configurations will exceed Mac OS, especially when adjusting the discovered bottlenecks. This approach did not explore any performance tuning in addition to the default installation guide provided by the official Docker documentation and the assignment of hardware resources. Finding the right testing platform to conduct fair experiments is a challenging task. It is important to note that using a nonofficial Linux distribution due to T2 chip limitation and installing a Windows machine through external software might affect the performance of tested configurations.

5.5. Potential Further Research

The results of the study showed large differences in performance between the tests. Extending the study conducted in this paper would help verify the reliability of the conclusions drawn. The large variation in the results between tests may indicate problems with the test platform presented. More research is recommended to minimize the influence of external factors.

5.5.1. Use Official Linux Distribution

With T2 chip installation of the official Linux system distribution did not succeed. One of the recommended expansions of this research could be replicating the tests with an official Linux distribution on an Apple computer without the T2 chip. This approach could focus on optimizing Docker installation, focusing on driver support, and addressing system-specific bottlenecks. The downside of this approach is that the T2 chip was introduced in late 2017. As a result, older machines must be used in tests when keeping an Intel-based MacBook as the test platform. This could lead to less relevant output.

5.5.2. Change Test Platform to Non-Apple

The different choices for a testing platform could benefit from more accurate Windows and Linux results. This approach would require preparing an Hackintosh machine [37], which would break the macOS EULA. Using this method could also lead to less reliable macOS performance output as Hackintosh is not an official distribution and is not supported. This approach may suffer from compatibility issues.

5.5.3. ARM Test Platform

With the growing popularity of the ARM architecture in personal computers, knowledge about the performance differences of Docker containers is desirable. All three tested operating systems do have an ARM distribution available. This approach could measure the performance of both ARM and x64 Docker images. This approach could be considered problematic, as there are many known issues for Docker Desktop for Mac with Apple silicon CPU [38].

5.5.4. Use More Than One Test Platform

The use of two testing platforms could bypass the difficulty of preparing a platform that fully supports all three operating systems. Two test instances with two installed systems, each including a common OS (preferably Linux), could determine the performance differences of all platforms. The drawback of this approach is greater exposure to external factors and greater complexity while analyzing test output.

5.5.5. Measure Performance Difference between Virtual Machine or Different Container Technology

Extending this study by adding an instance of a virtual machine as a third comparison point could result in a deeper understanding of the performance overhead between these two technologies and a bare metal operating system. Knowledge about performance differences between various container environments could determine which platform is better in terms of virtualization in general.

5.6. Summary

In essence, both the theoretical analysis and the experimental part of this research paper highlight the performance influence of the operating system and the Docker runtime. This complex relationship emphasizes the importance of careful and reasonable decisions, guided by a precise analysis process of all the given requirements. This study emphasizes how to decide on the performance difference that might be observed between popular platforms.

5.7. Future Work

In their future research, the authors will want to consider a more comprehensive/realistic benchmark that exercises CPU, I/O and network at the same time. In addition to more complex tests, the authors will also want to include in their research the latest operating systems such as MacOS 14 Sonoma and Windows 11. An additional field of study will be to include in the tests also the M1/M2 based Mac computers.

Author Contributions: Conceptualization, D.K. and M.S.; methodology, D.K.; software, D.K.; validation, D.K.; formal analysis, D.K.; investigation, D.K.; resources, D.K.; data curation, D.K.; writing—original draft preparation, D.K.; writing—review and editing, D.K. and M.S.; visualization, D.K.; supervision, M.S.; project administration, M.S.; funding acquisition, M.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Polish Ministry of Science and Higher Education (No. 0313/SBAD/1311).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPU	Central Processing Unit
GPU	Graphics Processing Unit
OOME	Out of Memory
PID	Process Identification

References

1. Docker Home Page. 2023. Available online: https://www.docker.com (accessed on 28 May 2024).
2. Stackoverflow.com Developers Survey 2022. 2022. Available online: https://survey.stackoverflow.co/2022/ (accessed on 28 May 2024).
3. Gholami, S.; Khazaei, H.; Bezemer, C.P. Should you Upgrade Official Docker Hub Images in Production Environments? In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Madrid, Spain, 25–28 May 2021; pp. 101–105. [CrossRef]
4. Best OS for Docker: Deciding Factors for Choosing Docker Host OS. 2023. Available online: https://www.knowledgehut.com/blog/devops/best-os-for-docker (accessed on 28 May 2024).
5. Sergeev, A.; Rezedinova, E.; Khakhina, A. Docker Container Performance Comparison on Windows and Linux Operating Systems. In Proceedings of the 2022 International Conference on Communications, Information, Electronic and Energy Systems (CIEES), Veliko Tarnovo, Bulgaria, 24–26 November 2022; pp. 1–4. [CrossRef]
6. Kithulwatta, W.M.C.J.T.; Jayasena, K.P.N.; Kumara, B.T.G.S.; Rathnayaka, R.M.K.T. Performance Evaluation of Docker-Based Apache and Nginx Web Server. In Proceedings of the 2022 3rd International Conference for Emerging Technology (INCET), Belgaum, India, 27–29 May 2022; pp. 1–6. [CrossRef]
7. Lingayat, A.; Badre, R.R.; Kumar Gupta, A. Performance Evaluation for Deploying Docker Containers on Baremetal and Virtual Machine. In Proceedings of the 2018 3rd International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 15–16 October 2018; pp. 1019–1023. [CrossRef]
8. Yadav, R.R.; Sousa, E.T.G.; Callou, G.R.A. Performance Comparison Between Virtual Machines and Docker Containers. *IEEE Lat. Am. Trans.* **2018**, *16*, 2282–2288. [CrossRef]
9. Xu, P.; Shi, S.; Chu, X. Performance Evaluation of Deep Learning Tools in Docker Containers. In Proceedings of the 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM), Chengdu, China, 10–11 August 2017; pp. 395–403. [CrossRef]
10. Zhao, N.; Tarasov, V.; Albahar, H.; Anwar, A.; Rupprecht, L.; Skourtis, D.; Paul, A.K.; Chen, K.; Butt, A.R. Large-Scale Analysis of Docker Images and Performance Implications for Container Storage Systems. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 918–930. [CrossRef]
11. Dordević, B.; Timčenko, V.; Lazić, M.; Davidović, N. Performance comparison of Docker and Podman container-based virtualization. In Proceedings of the 2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, Bosnia and Herzegovina, 16 March 2022; pp. 1–6. [CrossRef]
12. Sun, Y.; Safford, D.; Zohar, M.; Pendarakis, D.; Gu, Z.; Jaeger, T. Security Namespace: Making Linux Security Frameworks Available to Containers. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1423–1439.
13. Stan, I.M.; Rosner, D.; Ciocirlan, S.D. Enforce a Global Security Policy for User Access to Clustered Container Systems via User Namespace Sharing. In Proceedings of the 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), Bucharest, Romania, 11–12 December 2020; pp. 1–6. [CrossRef]
14. Unshare-Linux Manual Page. 2023. Available online: https://man7.org/linux/man-pages/man1/unshare.1.html (accessed on 29 May 2024).
15. The Linux Process Journey—PID 1 (Init). 2022. Available online: https://medium.com/@boutnaru/the-linux-process-journey-pid-1-init-60765a069f17 (accessed on 29 May 2024).
16. Docker Run Reference. 2023. Available online: https://docs.docker.com/engine/reference/run/#foreground (accessed on 29 May 2024).
17. Biradar, S.M.; Shekhar, R.; Reddy, A.P. Build Minimal Docker Container Using Golang. In Proceedings of the 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 14–15 June 2018; pp. 999–1000. [CrossRef]
18. PS- Linux Manual Page. 2023. Available online: https://man7.org/linux/man-pages/man1/ps.1.html (accessed on 29 May 2024).
19. Nickoloff, J.; Kuenzli, S. *Docker in Action*, 2nd ed.; Manning: Amazon UK, 2019.
20. Mount Namespaces-Linux Manual Page. 2023. Available online: https://man7.org/linux/man-pages/man7/mount_namespaces.7.html (accessed on 29 May 2024).

21. Yang, T.; Luo, Z.; Shen, Z.; Zhong, Y.; Huang, X. Docker's Security Analysis of Using Control Group to Enhance Container Resistance to Pressure. In Proceedings of the 2019 10th International Conference on Information Technology in Medicine and Education (ITME), Qingdao, China, 23–25 August 2019; p. 656. [CrossRef]
22. Runtime Options with Memory, CPUs, and GPUs. 2023. Available online: https://docs.docker.com/config/containers/resource_constraints/ (accessed on 29 May 2024).
23. Gorman, M. *Understanding the Linux Virtual Memory Manager*; Prentice Hall PTR: Hoboken, NJ, USA, 2004.
24. Petazzoni, J. *Cgroups, Namespaces and beyond: What Are Containers Made from?* Docker: Barcelona, Spain, 2015.
25. What Is WSL? 2023. Available online: https://learn.microsoft.com/en-gb/training/modules/wsl-introduction/what-is-wsl (accessed on 29 May 2024).
26. Windows 11 WSL2 Performance Is Quite Competitive Against Ubuntu 20.04 LTS/Ubuntu 21.10. 2021. Available online: https://www.phoronix.com/review/windows11-wsl2-good (accessed on 2 June 2024).
27. VPNkit Source Code Repository. 2023. Available online: https://github.com/moby/vpnkit (accessed on 2 June 2024).
28. Under the Hood: Demystifying Docker Desktop for Mac. 2018. Available online: https://collabnix.com/how-docker-for-mac-works-under-the-hood/ (accessed on 2 June 2024).
29. Software License Agreement for macOS Ventura. 2022. Available online: https://www.apple.com/legal/sla/docs/macOSVentura.pdf (accessed on 2 June 2024).
30. Arm-Based PCs to Nearly Double Market Share by 2027. 2023. Available online: https://www.counterpointresearch.com/arm-based-pcs-to-nearly-double-market-share-by-2027/ (accessed on 2 June 2024).
31. Sladović, D.; Topolčić, D.; Delija, D. Overview of Mac system security and its impact on digital forensics process. In Proceedings of the 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 28 September–2 October 2020; pp. 1236–1241. [CrossRef]
32. T2 Ubuntu Repository. 2023. Available online: https://github.com/t2linux/T2-Ubuntu (accessed on 2 June 2024).
33. Install Windows 10 on Your Mac with Boot Camp Assistant. 2020. Available online: https://support.apple.com/en-asia/HT201468 (accessed on 2 June 2024).
34. Advanced Settings Configuration in WSL. 2023. Available online: https://learn.microsoft.com/en-us/windows/wsl/wsl-config#configure-global-options-with-wslconfig (accessed on 2 June 2024).
35. Kovács, A. Comparison of different Linux containers. In Proceedings of the 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, Spain, 5–7 July 2017; pp. 47–51. [CrossRef]
36. Benchmarking MySQL 5.7 Using Sysbench 1.1. 2023. Available online: https://minervadb.com/index.php/2018/03/13/benchmarking-mysql-using-sysbench-1-1/ (accessed on 2 June 2024).
37. Hackintosh-Project Site. 2023. Available online: https://hackintosh.com/ (accessed on 2 June 2024).
38. Docker Documentation: Known Issues for Mac with Apple Silicon. 2023. Available online: https://docs.docker.com/desktop/troubleshoot/known-issues/ (accessed on 2 June 2024).