IMPACT OF THREAD VIRTUALIZATION ON DOCKER COMPARED TO NATIVE SYSTEM

by Harsh Nagarkar

Dec 2020

ABSTRACT

Most modern day computing infrastructure is in some way related to cloud. Building a scalable system, load balancing, traffic management are just some of the things done in the cloud. These compute heavy multiprocessor and multi threaded tasks require a secure platform, flexibility of infrastructure, availability of resources along with necessary isolation per service to function properly. These design challenges are solved by virtualization technologies like virtual machines and containers.

The purpose of this paper is to evaluate the performance of threads running inside Docker container and native machine environment. Three programming languages Python, C and Go, each were used to write matrix multiplication, pi calculator and DNS lookup resolver threaded program. These programs were then benchmarked with respect to the time. After run time comparison, it was concluded that interpreted languages like Python performed differently compared to compiled languages like C and Go due to more noticeable overhead. In compute heavy tasks, matrix multiplication, Go native was the fastest performer, where as in synchronization tasks, calculating pi, C was fastest performer and had similar measurements in both of the environments. Finally in in I/O bound tasks, DNS resolver, the C was the fastest performer and had similar measurements in both of the environments. Based on all of the measurements it cannot be generalized that Docker is better for threading than the native systems or vice or verse, but it can be concluded that Docker adds more noticeable overhead and also speeds up the program in some cases by providing necessary isolation. Native system sometimes performs better in compute heavy tasks by providing easier access to resources. Hence the thread virtualization's performance is not only impacted by the program's environment but by also each programming language and program's interaction with the hardware.

ACKNOWLEDGEMENTS

Thanks to the professor, who helped me to reach here.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Building scale-able systems and launching clusters of computers in the cloud is the trend of modern day computing.This has started a new cloud computing era, where we have SAAS (software as a service), PAAS(plateform as a service) and IAAS (infrastructure as a service).[2] In past years bare metal servers were the only way to run programs in cloud. Emergence of new technology like virtualization has completely changed the bare metal server market and has enabled software to run in the virtual environment. Virtualization provides flexibility in choosing size, adaptation to changing capacity, increment or decrements in processing power, growth on demand, and energy efficiency to each of the applications running inside the virtual environment. [3][4]

Many tech companies like Amazon, etc use virtualization technologies to handle huge workloads in their data centers.[2] Linux, Windows are among many other operating systems that support these technologies. These technologies provide the necessary isolation from the host operating system and thus allow multiple operating systems to run multiple applications on the same physical server. This also provides security and platform independence from the host operating system in the program. One of the major benefit of virtualization is isolation of hardware resources. This allows each virtualization to either have shared resources, or have their own small or large pool of resources.[3].Eg: Kubernetes,etc. This resource separation carries some additional overhead which can slow down the performance of the application but it is worth the cost in most cases.

The purpose of this paper is to compare the threading performance impacted by virtualization technologies to native system. As there are many virtualization technologies

out in the computing market, selecting the most appropriate one is a challenging task. Some of the open sources and private example are Xen, LXC, Libvmi, Vmware, AWS fargate, Google Kubernetes Engine, etc. For the purpose of this study, Docker, a containerization program is chosen. Docker(PAAS) allows desired operating system to run in its own private name space. Therefore the program inside Docker has to communicate to the host operating system for resources.The technology is currently supported by Windows and Linux operating system. [1][5]

This paper uses Docker, a virtualization technology to run three programs namely matrix multiplication calculator, Monte Carlo Pi calculator, and DNS look up resolver. Each of these programs are written in Python, C, and Go programming language independently. Results were plotted after benchmarking the performance of these programs and the findings were researched.

CHAPTER 2

Virtualization and Native Technology

Virtualization refers to abstraction of computer resources.[1]. The virtual systems operate like real physical server like systems.[6]. Many companies like Google, Amazon, rackspace provide these servers out for rent.[2] These instances can be configured with varying amount of ram and CPU cores. This allows creation of a small single core system with a couple megabytes of ram to a full fledged back end server. Hence virtualization is primarily used to create clusters of distinct size by combining virtual instance systems into a single or multiple group.

Virtualization is essentially divided into two main categories, container based virtualization and hyper-visor based virtualization.

2.1 Hypervisor based virtualization

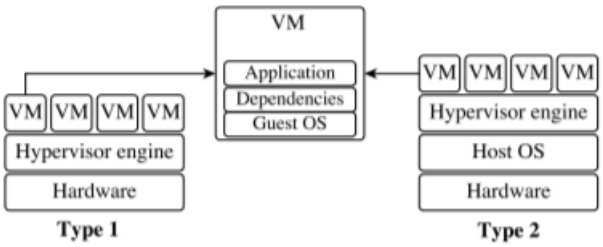

Figure 2: Architecture of Hypervisor-based Virtualization

Figure 2.1: Hyper-visor based virtualization design

The hypervisor based virtualization establishes virtual machines on top of the host operating system as shown in Figure 2.3. Each virtual machine comprises of the guest

operating system(OS), which holds application and it's dependencies and runs on the host operating system. Hypervisor based virtualization is further divided into two types.

2.1.1 Type 1 hypervisor or Bare metal or full virtualization

, In this virtualization, the virtual machine has direct access to hardware, therefore there is no need of a host OS. Eg:- KVM or kernel virtual machine uses 'QEMU' to emulate hardware access for guest OS, hyper-v is windows version of type-1 hypervisor.[6] Fig 2.3

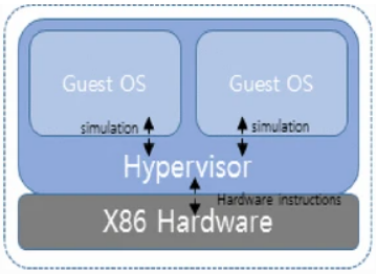

Figure 2.2: Hyper-visor based virtualization design full [1]

2.1.2 Type 2 hypervisor or para virtualization or hosted hypervisor

In this virtualization the guest OS sits on top of host operating system so the hardware can be accessed by accessing the host OS system calls. eg: Vbox, Vmware,etc. This virtualization is heavy as it includes the full fledged guest operating system communicating to host OS, and thus has a lower density of application packing and slightly slower performance.[6]

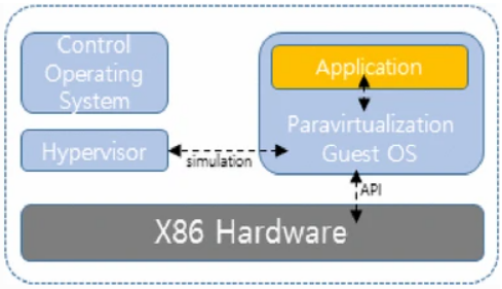

Figure 2.3: Hyper-visor based virtualization design para [1]

2.2 Container based virtualization

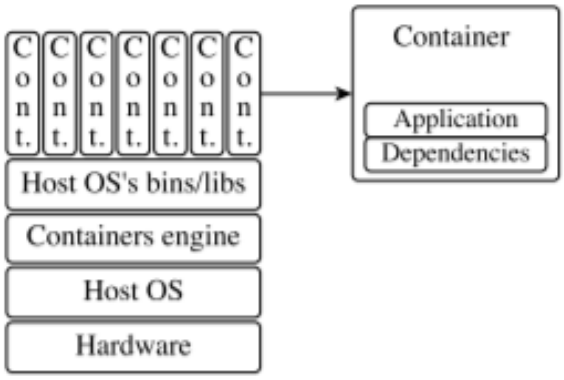

Figure 2.4: Containerized virtualization [1]

Container based virtualization is a light weight virtualization approach which allows running multiple virtual operation system environments on a single machine. These environments are called containers. Container based virtualization virtualizes at the operating system level thus allowing multiple applications to operate without redundantly running other OS kernels on the host. The containers appear to be normal processes from host view. These containers run on top of a kernel, sharing the host machine. Hence containers get isolated resources to execute application, which could be shared with host or used separately. [6]

2.2.1 Docker

Docker container was a platform which used to be based on LXC container. This platform currently uses lib container for containerization. Docker takes advantage of four things to make containerized ecosystem work namely name spaces, cgroups, union file system and docker image. A name space wraps the operating system resources into different instances so that they can be isolated from the rest of the processes and used separately by containers. A limit can be set on them. Control groups provides a mechanism to account

and limit resources. Union file system allows combining different file systems into one file system for the Docker. Docker image is the Linux image that needs to be run to execute OS. All these components work in sync and use the underlying host operating system to communicate with hardware to schedule resources. Thus there is this, docker engine's extra abstraction layer between container and host OS.

CHAPTER 3

Implementation

The primary task of this evaluation was to accurately benchmark the three pre-selected programs; matrix multiplication, pi value calculator(Monte Carlo algorithm) and DNS resolver to draw out inferences. Each of these programs was written in three programming languages; Python, Go and C keeping the code functionality and complexity similar.

3.1 Algorithms

3.1.1 Matrix Multiplication

Matrix multiplication

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1} \\ a_{2,n+1} \\ \dots \\ a_{n,n+1} \end{bmatrix}$$

In matrix multiplication each row is multiplied by each column and then all entries are summed up leading to a single output matrix cell value in a separate thread. Hence the number of total cells in output matrix would be equivalent to number of threads. Two matrices of 100x100 size were used in this study. This creates 10000 threads for the program.The algorithm is fast as their is no sharing of data.

3.1.1.1 Language differences

The C code was modified and transliterated into other languages. [7] An important thing to note is Python and Go support objects but C does not. Therefore C implementation

uses array of threads whereas in Go a wait group and in Python a thread list was used.

3.1.2 Pi value calculator(Monte Carlo)

In Monte Carlo Pi value approximation calculation, random points are plotted on a canvas containing a unit circle embedded in a unit square. The points inside the circle divided by total points multiplied by 4 gives the approximate pi value as shown in Equation 3.1 where 'P' stands for points.

$$\pi = 4 * \frac{P_{\text{circle}}}{P_{\text{total}}} \tag{3.1}$$

The points are distributed to each thread by dividing the total points by number of threads as shown in Equation 3.2 where 'P' stands for points and 'T' for threads. In the study 3 million points were chosen to be distributed between 30 threads.

$$P_{\text{perThread}} = \frac{P_{\text{total}}}{T_{\text{count}}} \tag{3.2}$$

3.1.2.1 Language differences

The C code was modified and transliterated into Python and Go. [8] In C we have created an array where as in Go we have waitgroup and list in python. We are making the assumption that random number generation between -1 to 1 will approximately take same time between different languages.

3.1.3 DNS resolver

DNS resolver is divided into two parts requester and resolver. The requester is fed up 5 files, a file per requester thread, each containing twenty to twenty five domain names. These domain names were parsed and then passed into the resolver to resolve. After resolving, the first IP address was written in the text file along with the linked domain name. There are 5 resolvers in part two. In this fashion one hundred and twenty two domains names were processed.

3.1.3.1 Language differences

The C code was modified and transliterated into Go and Python.[9] Because C is relatively old language, in requester there are 2 mutex locks and in resolver there are three. Whereas in Go and Python implementation we just needed one mutex lock as Python queue and Go channels used are thread safe.

3.2 Infrastructure setup

Two virtual machines with eight cores and eight gigabyte ram, each hosted on Google cloud platform running Docker and Ubuntu were used. The native system's evaluation results were drawn by running the bare virtual machine with programming language environment. While the Docker evaluation results were drawn out the same way but in language specified containerized system.

3.3 Sample generation

Multiple measurements for each of these programs were taken using bash 'for' loop and then piped out into the text file for further calculation. To choose a sample size, the following formula was used.

$$n = (\frac{z_c * \sigma}{E})^2 \tag{3.3}$$

In the equation 3.3 the confidence value zc was set to 99% which is equivalent to 2.78 and Error was set to 0.2, 0,3 and 0.8 each for matrix multiplication, pi value calculation and DNS resolver respectively. Based upon the selected n ranged sample set, the average value was calculated.

3.4 Plotting

Matplotlib in python was used to plot the resultant bar graphs. These figures will later guide the research to discover the factors contributing to the measurement of performance.

CHAPTER 4

Performance Evaluation

4.1 Programming Languages

4.2 Case study

This paper uses three algorithms each written in Python, C and Go.

4.2.1 Matrix multiplication

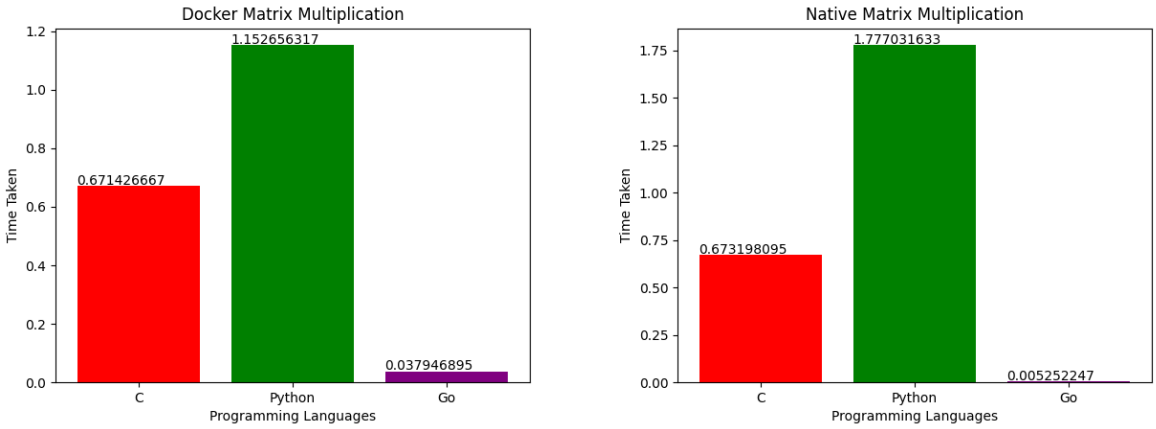

Figure 4.1: Matrix multiplication implementation

In matrix multiplication native system beats Docker in Go programming language, but Docker wins in Python. C programming language performs similar in both the cases. Figure 4.2.1. Go language finishes close to 100 times faster when compared to C.

4.2.2 Calculating pi

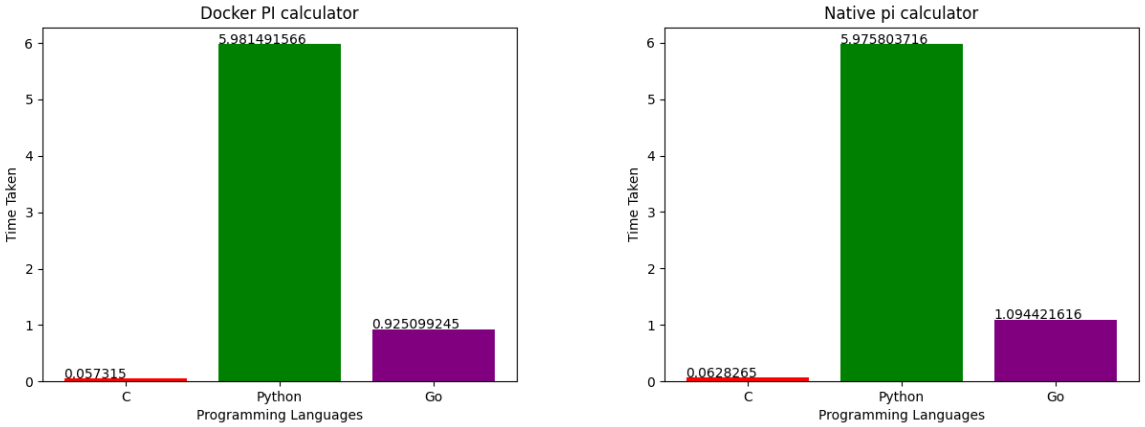

Figure 4.2: Monte Carlo Pi value implementation

In Monte Carlo pi approximation value calculation, we can observe from Figure 4.2.2 that native run time is slower as compared to Docker in case of Go programming languages, although in case of Python and C the reading are similar considering the margin of error. Also Python program took significantly longer than Go and C.

4.2.3 DNS resolver

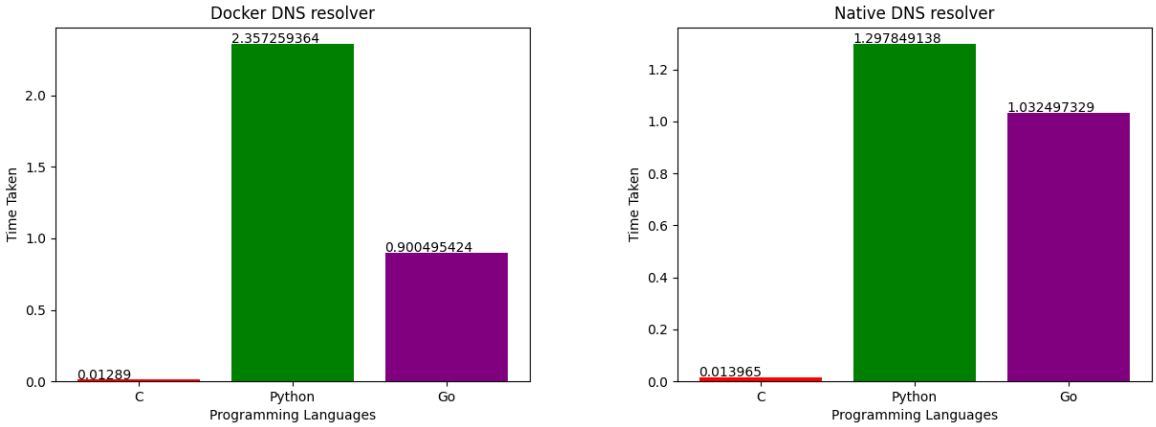

Figure 4.3: DNS resolver implementation

In Go program, native is slower than Docker-ized version. Although in case of Python Docker is slower than native system. C shows similar readings in both and out performs others programming languages.

CHAPTER 5

Discussion

5.1 Programming Languages

The results showcase's that C is the fastest programming language for pi calculation algorithm. This is due to it's transparent nature between the host operating system and the program caused by portable operating system interface(POSIX) standard threading library, pthread. [10] Python takes longer to complete in multi threading workloads compared to C and Go because it is an interpreted language. Python uses the system pthreads under the interpreter layer. As Python's thread manager is integrated in the interpreter itself, the slow performance of this language is caused by GIL or global interpreter lock during multi threading. The GIL's job is to transfer the control of thread to thread manager after every certain byte code, so that thread manager can decide which thread has the control for the next time. Thus only one thread has access to the interpreter at once. This means that only one thread will be in state of execution at once, even when multiple threads are scheduled and running on multiple processing cores. This is the bottle neck of python especially in CPU bound processes. [11]. The Go programs performed between run time of Python and C, except in case of matrix multiplication when the Go finishes execution before anyother language. Go routines are go function designed to run Go tasks in asynchronous order. Go routines are lighter than pthreads and may or may not be multiplexed on separate threads. [12]. This is why they are faster and lighter than pthreads.

5.2 Docker VS Native comparison

The data trend is not consistent enough to make a generalized conclusion hence case by case analysis is required. It should be noted that Go and C are binary packages

where as python is an interpreted language. Binary execution will always be faster than the interpreted binary packages. Each language and program specific Docker container is optimized for chosen specific language and program. This means that all extra libraries which are not related to that language or program are removed.

5.2.1 Matrix Multiplication

Evaluation confirms that the native system is faster than Docker for matrix multiplication program for the Go programming language. These observations are a result of reduced overhead during system calls in the native system compared to programs running from inside the Docker container. Go program turned out to be the winner as the Go routines are faster for execution than pthread as mentioned earlier in this paper. C showed the similar performance in both of the environment systems.

Python performs better in Docker container compared to native. This might be caused due to less competition for resources when context switching between thread functions. Indirectly this is due to separate inter process communication (IPC) name space and separate resource group which Docker provides. This leads to a better access time for GIL switching.

5.2.2 Calculating pi

Evaluation confirms that Docker is faster than native, in case of Go programming languages. This might be due to IPC name space group isolation which provides better computation and easier mutex lock access. This creates more positive impact than additional overhead incurred due to Docker. C program showed similar performance in both of the environments.

Python shows the opposite behavior where it appears to be faster in native. Although when we consider the error which is 0.2 the performance is similar on both of the environments.

5.2.3 DNS resolution

Majority of DNS resolution program is composed of I/O bound tasks related to networking and file access. Hence this program can be classified as I/O bound program. Evaluation confirms that Go program is slower in native compared to Docker. C program shows similar performance in both Docker and native environment. This is due to significant large error +/- 0.08 and C program's run time measurement of close to 0.02. The error was set high as a result of high noise. Python program in Docker is slower than native due to overhead caused by name space isolation for IPC, mutex lock access, networking and file access. This lead to a increase in latency.

In the case of Python, Docker is slower than native system. It appears that interpreted language gets affected significantly due to additional overhead caused by Docker's abstraction layer. It can still be concurred that the performance of Python is nearing Go program's measurement as I/O bound thread do not participate in GIL mechanism and can process independently.

5.3 Source for possible errors

Some assumptions were to made when bench marking the matrix multiplication, pi calculation and DNS resolver programs. These assumptions can cause errors in measurements.

It was assumed that the random number generator producing a number between negative one and one would take approximately same time in all languages. Yet the pseudo random number generator for three languages were not the same. This can cause a small shift in pi calculator program which might lead to error.

All language programs were designed to behave identically although there were few a differences between each due to language limitation. These differences lead to addition of extra mutex locks and usage of different data structure. Even so the overall effect of these changes was minimal on the result.

As Google cloud platform was used for tests, Googls's DNS server were used by virtual machines. The DNS lookup was thus assumed to be generalized and uniform when it can be date time traffic dependent. This can cause a small shift in measurement, leading to error.

The disk that had the code for the program was mounted on the docker system. Hence when there was a write the system call needed to be passed from Union file system to ext4 fie system for Ubuntu. Hence there was small lag when accessing the files during the write operation of the DNS resolver.

The underlying virtual machine running the native system was a full fledged Ubuntu machine used for bench marking. This machine also scheduled and ran tasks unrelated to the benchmarking program example:- OS tasks,etc unlike Docker where the software is optimized for the program running. This adds up a small random nature to the program's benchmark in the native system.

CHAPTER 6

Conclusion

Three programs, matrix multiplication, pi calculator and DNS resolver written in three different languages Python, Go and C were ran on a Docker and a native system. After benchmarking these programs the results were plotted using "matplotlib" and then investigated. Matrix multiplication or jobs that run asynchronously without shared data ran the best in Go on native system, followed by C in both environments. Pi value calculation or programs that have shared data ran the best in C where either Docker or native performed similar where as Go in Docker performed slightly better due to reduced noise in the system. DNS resolver or programs which were I/O bound ran the best in C where either the Docker or native performed similar where as Go was faster in Docker compared to native. As the overhead in interpreted language is significantly larger, Python was the slowest language amongst all. Python's native environment performance is faster than Python's Docker performance in DNS resolution or I/O bound program due to less overhead and faster access time. Python programs showed similar performance in both environments in pi calculation program. CPU bound tasks like matrix multiplication were slower in Python native environment compared to Docker.

Docker uses Cgroups, Union Files System, name spaces and an OS image to create a containerized environment where as native system uses plain operating system to create a individual separate environment. After inspecting findings discussed above, native system performs better in CPU bound threaded tasks. If the program involves synchronization, then both environment show similar results except in case of Go where Docker is faster. Finally, if the program has more I/O bound tasks like file system access and networking then C

performed faster and similar in both of the environments otherwise Docker is the better fit for Go programming and Native is better fit in python programming. It cannot be concluded that Docker is better than native or vise of versa but it can be concluded from the findings that the Docker because of it's design adds a noticeable overhead when running medium to large jobs but also provides desired separation form the host to reduce noise and speed up computation. Native system's performance was faster in some programs due to fast access time to reach resources. This was achieved as a result of less overhead and easier OS program interaction. Hence performance of thread virtualization can be positive or negative based not only upon the Docker or native environment but also upon the programming language and program's interaction with the hardware.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] M. Chae, H. Lee, and K. Lee, "A performance comparison of linux containers and virtual machines using docker and kvm," *Cluster Computing*, vol. 22, no. 1, pp. 1765–1775, 2019.

[2] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2015, pp. 171–172.

[3] N. G. Bachiega, P. S. Souza, S. M. Bruschi, and S. D. R. De Souza, "Container-based performance evaluation: A survey and challenges," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 398–403.

[4] G. Bhatia, A. Choudhary, and V. Gupta, "The road to docker: a survey," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 8, 2017.

[5] P. R. Desai, "A survey of performance comparison between virtual machines and containers," *ijcseonline. org*, 2016.

[6] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[7] Macboypro, "Matrix multiplication in [c] using pthreads on linux," May 2009. [Online]. Available: https://macboypro.wordpress.com/2009/05/20/matrix-multiplication-in-c-using-pthreads-on-linux/

[8] M. Ballantyne, "michaelballantyne/montecarlo-pi," Oct 2012. [Online]. Available: https://github.com/michaelballantyne/montecarlo-pi/blob/master/pi.c

[9] J. Jacobsen, "jjjacobsen/multi-threaded-dns-lookup," Apr 2017. [Online]. Available: https://github.com/jjjacobsen/multi-threaded-DNS-lookup

[10] B. Barney, "Posix threads programming," *National Laboratory. Disponível em:¡ https://computing. llnl. gov/tutorials/pthreads/¿ Acesso em*, vol. 5, p. 46, 2009.

[11] N. Matloff and F. Hsu, "Introduction to threads programming in python," *University of California*, 2005.

[12] R. Prabhakar and R. Kumar, "Concurrent programming with go," Citeseer, Tech. Rep., 2011.