International Journal of Computer Applications (0975 - 8887)
Volume 176 - No.36, July 2020

Modeling Performance and Scalability of Cloud Services over Unikernels

Wagner S. Marques
Federal Institute Farroupilha
Alegrete - Brazil

Arthur F. Lorenzon
Federal University of Pampa
Alegrete - Brazil

Marcelo C. Luizelli
Federal University of Pampa
Alegrete - Brazil

Fabio D. Rossi
Federal Institute Farroupilha
Alegrete - Brazil

ABSTRACT

Virtualization provides several benefits to computing. However, applications have presented a low performance on traditional virtualized platforms. Unikernels are an alternative for cloud platforms application deployment in order to overcome such limitations. It comprises a small system for the cloud, providing deployment agility and portability among virtualization platforms. Moreover, Unikernels performance evaluation concerning other virtualization alternatives such as containers and traditional virtual machines is still lacking. This work evaluates Unikernels' performance and scalability running HTTP services. The data was collected through a benchmark tool execution and modeled using a polynomial and a linear equation. The results illustrate that Unikernels is a promising alternative since it presents a 34.9% better performance than traditional virtual machines.

General Terms

Algorithm, Network

Keywords

Unikernels; Containers; Virtualization; HTTP; Web application; Cloud

1. INTRODUCTION

Virtualization technology provides several benefits to computing, such as equipment acquisition costs reduction and enhancing computer resources managing. This technology consists in to execute some operating systems (OS) simultaneously on the same host, where each system, which is called a virtual machine (VM), is presented as an autonomous system, starting a kernel instance and running several applications. Each of these instances is specialized and used for a specific purpose, such as a web server or a database [8].
However, the applications have a lower performance on VMs than on physical machines. The overhead generated by the virtualization layer (hypervisor) is one of the main concerns when virtualized environments are adopted. In this sense, identifying and reducing such cost has been several study research subjects [6] [13]. In order to address such limitations, container virtualization was proposed. Unlike traditional virtualization, it does not implement a kernel for each virtualization instance and does not have a hypervisor for running VM. Instead, such technology focuses on the performance of applications (at the price of security and isolation), considering that all instances running on the same kernel [16].
Currently, a new proposal of virtualization arises, focusing on the need for more security and computational performance: Unikernels. A Unikernel is a specialized VM that eliminates the operating system layer and provides a minimal attack surface and a quick booting [14]. This new type of VM is based on operational library systems, and its purpose is offering smaller, faster, and more secure systems, optimized for a single application. These characteristics make Unikernels suitable to restricted environments that aim for the lowest computational resources cost and use, such as clouds [10].
Unikernels are single address space systems that enable a selection of system component bundle for a specific application into a single system image. Such a simple image can be run on the cloud or directly on the hardware, depending on what driver components are available for that particular Unikernel system adopted during development. Unikernels can be implemented in two ways: (i) Unikernels use single languages such as Erlang, Haskell, or OCaml, and a clean-slate implementation of everything, and (ii) Unikernel executes the software which runs on current operating systems and servers, such as Apache2, Haproxy, and Nginx.
This work proposes Unikernels performance and scalability evaluation (Mirage/Rumprun) concerning other virtualization alternatives, such as traditional virtual machines (Xen), and containers (Docker) running HTTP services. It also evaluates Unikernels' behavior over two different load balancers. It presents two equations in order to predict the impact on the performance when some Unikernels are adopted over the same host. In this way, this work creates a polynomial function to express the trendline of the transfer rate during the Unikernels numbers increase. It also illustrated the results applied in the Linear progression to represent the relationship between Unikernels' performance and the number of vCPU available on the host.
The results were considered promising from Unikernels since they presented a better performance compared to traditional virtualization using the Xen hypervisor. The containers still present a better performance than Unikernels. This paper is organized as follows:

Section 2 presents a background showing fundamental applications and technologies concepts. The evaluation and results are discussed in Section 3. Section 4 discusses some related works. Section 5 brings conclusions and future work.

2. BACKGROUND

Unikernels are specialized VM compiled into a single and minimal system image based on operating system libraries, runtime language, configuration, and application files. They are bootable on a hypervisor such as Xen or KVM [14]. The OS library aims that entire customization of the OS on which an application depends runs in its address space as a library [12]. Besides, when the Unikernels are executing on a standard hypervisor, it avoids hardware incompatibility found when is used a traditional library system. In the same context, it is structured differently compared to the conventional operating system because it is implemented with all services and libraries linked to the application. Precisely, the application is related to several libraries that provide a set of high-level abstractions of the operating system, such as processes and files [8]. Figure 1 presents the difference between traditional and Unikernel virtualization architectures.

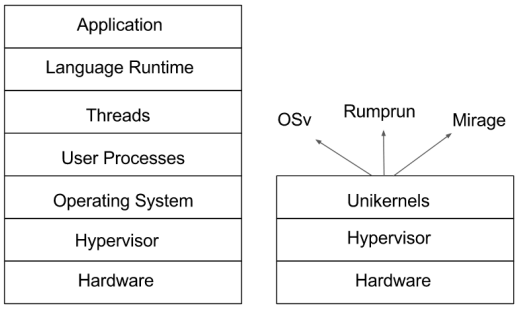

Fig. 1. Difference between traditional and Unikernel virtualization architectures.

The Unikernels premise is to have small file size and lower attack surface than conventional VMs. Unikernels' file image rarely exceeds 1 MegaByte (MB), and many situations accomplish their task using only 32 MB of memory with to almost instant initialization. Thus, it is possible to have thousands of Unikernels on a single physical host [14]. Besides treating the database or the web server as standalone applications such as traditional VMs, Unikernels treat it as libraries within a single application, allowing the developer to create them using simple library calls. Although there are many existing Unikernel systems currently and each of these applications was developed with specifics proposes.

2.1 Xen

Xen was developed in order to enable running multiple operating systems on a single server and to provide migration of operating system instances running among different servers. Such technology allows resources partitioning of a host and then providing resource isolation for VM. Besides, it also enables that multiple different operating system images can run in a parallel way on the same physical host. Xen is a type 1 hypervisor that is characterized by running directly on the hardware without the need for an operating system, unlike the type 2 hypervisor, which runs as software on the operating system. The difference between hypervisor type 1 and type 2 is only where it is presented. Otherwise, it has the same function [11]. Figure 2 shows the Xen hypervisor architecture.

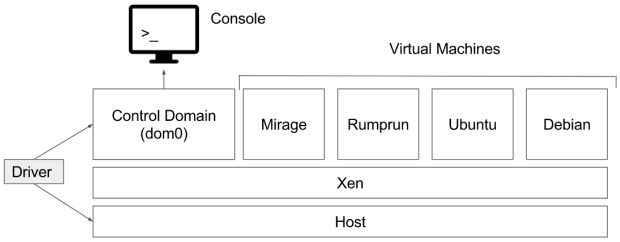

Fig. 2. Xen hypervisor architecture.

Xen architecture is divided into Control Domain (Domain 0) and Guest Domains (VMs). The Control Domain is a specialized layer that offers exclusive privileges such as the capability to access the hardware directly, running directly on the device, and it is responsible for managing CPU, memory, and interrupts. The Control Domain also provides an API that allows the management of all virtual machines. Also, several guest domains can be used simultaneously on a single hypervisor, such as Unikernels (Rumprun), and general-purpose systems (Ubuntu).

2.2 Docker

Containers provide a different level of abstraction and isolation compared to traditional virtualization. Whereas hypervisors virtualize hardware drivers for each VM instance, which generates overhead, containers avoid such overhead implementing a process of isolation at the operating system level. In other words, a container runs as an isolated process in userspace on the operating system, while the host operating system kernel is shared among all instances of containers. Then, the lack of driver virtualization, coupled with the use of a shared kernel, provides the capability to achieve higher performance and smaller disk images [9] [1].
Docker is an open-source platform. It introduces an underlying container mechanism and a functional Application Programming Interface (API) that makes it easy to build and to manage (*i.e.,* application removal). Within a Docker container, one or more processes and applications can be run concurrently. In this sense, since the Docker containers do not need the hardware and driver virtualization, coupled with the use of the shared kernel, it provides the ability to achieve higher performance and smaller disk images [9] [1].

2.3 Mirage

The MirageOS is an application that allows the Unikernels development to compile and link the code into a bootable Xen VM image. MirageOS uses the OCaml language with libraries that support network and storage and work under Unix during its development, but become OS drivers when compiled for deployment in production on a hypervisor. OCaml is an imperative and object-oriented system programming language adopted by presenting a flexible programming model. It is characterized by its brevity, which reduces counts of code lines that are often considered correlated with an attack surface in systems [8].
MirageOS is stable and usable, having reached release 3.0 in 2017. The project website[1] has proper documentation, a few samples, and

[1]MirageOS. Available at: <https://mirage.io>.

tutorials to help the new developers to start to build Unikernels. It also contains some tutorials and presentations that provide support to the new ones. Several MirageOS contributors have authored academic white papers, and the Docker now employs a significant MirageOS portion. Besides, Mirage includes clean, functional implementations of protocols ranging from TCP/IP, DNS, SSH, and HTTP.

2.4 Rumprun

Similar to Mirage, Rumprun is considered an alternative to containers, with more reliable isolation. It is based on rump kernels that provide free, portable, and componentized kernel drivers. Furthermore, this application allows the running of existing unmodified software (e.g., Nginx, haproxy) as a Unikernel [5]. Also, Rumprun supports multiple platforms, including bare metal and hypervisors such as Xen and KVM. In order to use various cores is a need to generate multiple Unikernels on the same host. This type of Unikernel can be created with an experimental toolchain to facilitate all processes of making.
Rump kernels use the NetBSD kernel architecture to provide unmodified NetBSD drivers. It avoids porting errors and gives the user the ability to choose the upstream drivers' vintage. In the same context, Rumprun is available under the BSD license, making it free for use. Unlike Docker, the application constructed to Rumprun does not depend on a full operating system (OS) installation and can be sent as self-contained with small-sized images. Since a rumprun image includes less code than an entire OS image, there is also less chance of shipping code with latent security lacks [5] [4].

3. MODELING AND RESULTS

This section presents the testbed used to feed the model, the modeling process, results, and discussion.

3.1 Testbed to feed the model

This work developed a simple static web page to evaluate the transfer rate on all of the virtualization alternatives adopted in this paper. To evaluate, this work used the Apache Benchmark (AB) tool. This application allows performing several HTTP requests to simulate many users simultaneously for a host in order to obtain the transfer rate of the service. In this sense, this work played 10.000 requisitions with 1.000 concurrent users. In the same context, all tests were performed on a host that comprises a CPU with 4 cores and 8 GB of RAM. The operating system used in the host, container, and Xen was the Debian Wheezy (release 8.8.0). This work also highlights that the Nginx HTTP server was applied over all of the virtualizations alternatives performed.
This work created a local Ethernet network to perform the evaluation and to avoid Internet connection latency. The requests using HTTP were delivered through another device connected to the same network of the machine defined as a server. In the server, was created bridged networking used to connect virtual machines to the external network using the host computer Ethernet adapter. Besides, the IPtables were used to connect VMs with the Internet and the other hosts on the same local network. IPTables is a Linux tool for firewalling and routing chains that perform port forwarding on a computer or server. The requests were executed by other equipment connected to the local network of virtualization system host. Such a network is illustrated in Figure 3.
Initially, it builts mirage Unikernels intended for HTTP service evaluations and compiled them into images for the Xen Hypervisor. However, such Unikernels presented a transfer rate of only 30

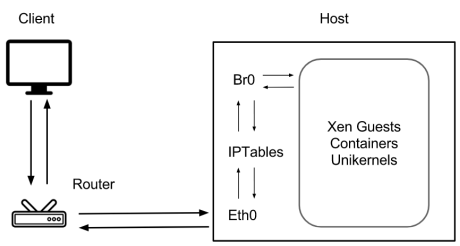

Fig. 3. Network connection configuration on our experiment

kilobytes per second. According to Briggs et al. [2], the Mirage TCP/IP library leaks memory every time a TCP connection is released, and it causes the Mirage performance losses. This error provides for Unikernels a lower result regarding performance when comparing to traditional virtual machines or containers. Unfortunately, this bug makes mirage Unikernels not be a suitable choice for HTTP services. This complete result is illustrated in Figure 6.

3.2 Modeling

This proposal created two equations in order to model the relationship between the Unikernels transfer rate performance and the Unikernels increase. To do this was adopted a polynomial regression. In the same context, it used linear regression to express the relationships among the performance and the number of vCPU units on the host. The features of this regressions are described below:

—Polynomial regression: It is commonly used in order to express the trendline of data that represents nonlinear results, and in our case, provide a Unikernel transfer rate estimation during the HTTP services execution.

$$y = b_1 + b_0.x + -20x^2$$

where y stands for transfer rate performance of Unikernels implanted and x for the number of Unikernels running on the server hosting the Xen hypervisor. The constants b0 and b1, the y-axis intercept, and they can vary among different applications.

—Linear Regression: In the same sense of the polynomial regression, this alternative can be used to apply a linear regression on the data to get a trendline and to predict the results. However, this solution is used when the collected data represent linear results.

$$y = b_1 * x + b_0$$

The same variables were adopted in this equation as well, where y presents the transfer rate performance and b1 and b0 the x-axis intercept. Although, the X represents the number of vCPU to Unikernels on the host or between different hosts through the sharing of computer resources to an application.

The trendline generated by a polynomial regression is illustrated in Figure 4. Highlights that just one Unikernel cannot use many vCPUs since it is a single-core system can be noted. Using this equation, one can see that it can be utilized eleven Unikernels, with the execution of the same application or with use of the different applications and services, without performance damage when compared to the use of just one Unikernel. The loss of performance occurs as the processor's resources are disputed among the Unikernels. One also can see a higher performance loss on every four Unikernels, and it happens because this Unikernel is competing for resources with the Domain0 that could be defined with just one vCPU as well.

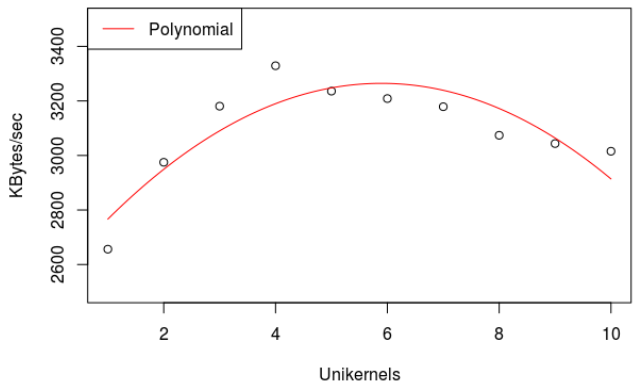

Fig. 4. Polynomial regression among performance and number of Unikernels

To demonstrate the performance concerning the increase of vCPU on the host, the linear progression trendline was chosen (shown in Figure 5). This equation was determined in order to provide a performance prediction of Unikernels with a different number of available vCPUs. It can be used to obtain the transfer rate with all Unikernels running on the same host or with these systems in different hosts through the use of a load balancer application. The trendline illustrates an increase in performance that occurs when are adopt just one image of the system for each vCPU, considering that our collected data have shown this increase when only one Unikernel is applied for each vCPU available. In this sense, it can be used to verify the adoption viability of Unikernels for a specific service or host.

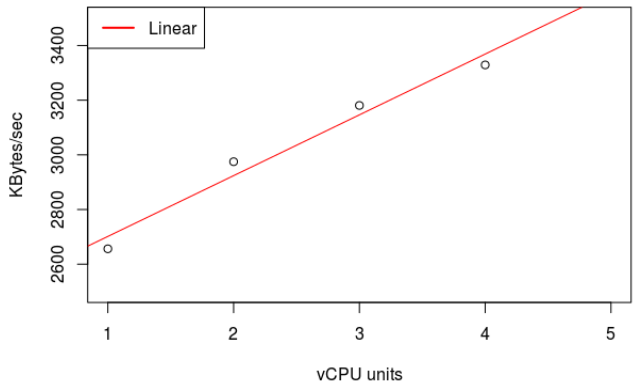

Fig. 5. Linear regression between performance and number of vCPUs available on the host.

3.3 Results and Discussion

Rumprun presented a 34.9% transfer rate better than traditional VMs over the Xen hypervisor. The build process was performed with an experimental toolchain and compiled to a Xen image bootable. This Unikernel, supported by Xen, is created and stays in a pause state. It will listen to the network waiting in a for a connection. When a connection exists, this Unikernel meets that user's demand and after returns to a pause state waiting for a new connection. Thus, the service cycle for connections is repeated. Even so, Unikernels have little startup time, managing to respond to users' connections faster than virtualized services in traditional environments such as Xen.

However, Unikernels still have lower transfer rates during the execution of HTTP services when compiled into containers. Docker presented a transfer rate of 46% better than Unikernels Rumprun, with very close application performance in the native system. It occurs because containers eliminate the virtualization layer and running as processes on to the host operating system. In this sense, these features make the containers perform better compared to Unikernels and traditional VMs. Besides, the system used in the containers was a version of Debian 8 modified for docker.

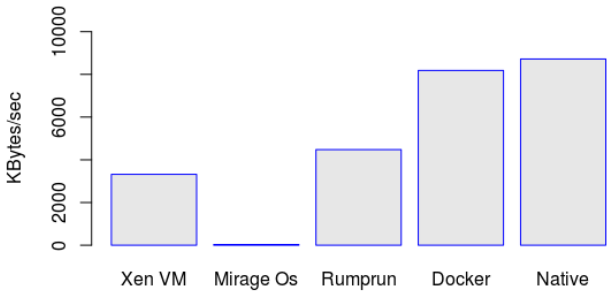

Fig. 6. Virtualization alternatives performance evaluation.

In the same context, Rumprun Unikernels are single-core, and in order to use all cores of the host processor, was developed and implemented a series of Unikernels on the same host. To do this was configured a load balancing application. For analysis purposes, was adopted two load balancer alternatives: Nginx and Haproxy. The Nginx is an open-source application commonly used as a web server, and it also can be used with a load balancer. Tha Haproxy is an open-source solution to load balancer as well. Although, it is focused on the single-purpose that is only to offer load balancing among applications. Both applications have the following load balancing algorithms:

—Round-robin: it is one simple method for distributing client requests across a group of servers. With the list of servers in the group, the round-robin load balancer forwards a client request to each server in turn.

—Leastconn: the next request is assigned to the server with the fewest active connections. The server selected with the least number of connections is recommended for longer sessions.

—Ip-hash or Source: it is a hash function that is used to determine what server should be selected for the next request based on the clients IP address. Also, it is one method to ensure that a user will connect to the same server.

The algorithm used in both applications was round-robin during all tests. The results indicate a significant improvement in the transfer rate when 4 Unikernels were used in both load balancing applications. It occurs because the host used has 4 processing cores. The results, presented in Figure 7, illustrated that with more than

4 Unikernels, the transfer rate decreases due to competition for processor resources. The hypervisor allows the processor sharing among the virtual machines through mechanisms that aim for efficient use of these resources. Each of the vCPUs (virtual Central Processing Unit) represents one physical processor unit.
In this sense, it is possible to perform the allocation of several virtual processors for virtual machines. Each of our Unikernels was implanted using just one of these vCPUs, and the Dom0 performed your tasks with only one vCPU as well. The transfer rate using Haproxy with 4 Unikernels is 20,2% better than when only one is used with a haproxy load balancer. In the same context, using the Nginx as the load balancer, 4 Unikernels presented 22,7% of improvement. The results that 4 Unikernels are 2% better using haproxy than when used the Nginx.

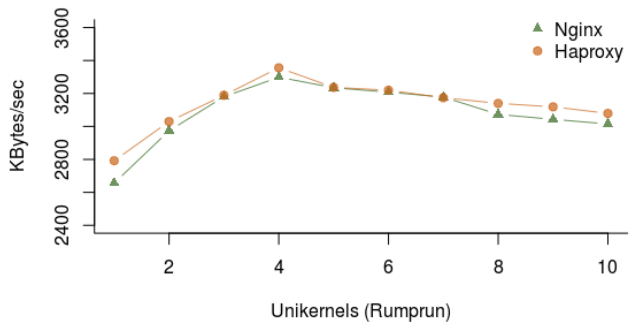

Fig. 7. Virtualization alternatives scalability evaluation.

However, the use of the load balancing tools generated an overhead in the transfer rate. Using only a Unikernel without a load balancing application, the results have 33% better performance than our best result with 4 Unikernels using the same applications. This result is because the algorithms use health checks to determine if a backend server is available to process requests. This feature aims to establish a TCP connection to the server and checks if the backend server is running on the configured IP address and port. In this sense, the overhead is generated.
Although Unikernels have a lower transfer rate than containers and applications on a native host, Unikernels are a promising virtualization alternative. When one considers that security is paramount, Unikernels presents as the best alternative since they have a better transfer rate than traditional VMs and have a minimal attack surface. Besides, the Unikernels can be used in embedded devices with virtualization support (e.g., CubieBoard 2) that in the Internet of Things frequently perform critical tasks, and the security is essential.

4. RELATED WORK

Several papers have developed and evaluated the performance of Unikernels in clouds. Xavier et al. [15] performed the spawning of several instances in order to investigate the impact on provisioning time. The authors chose the OSv project as the Unikernel representative since it supports different hypervisors and language runtimes. The Docker and KVM were the other virtualization alternatives wanted during the evaluation. The results were obtained on multiple instances concurrently in an OpenStack cloud platform and indicated that OSv outperforms the alternatives.
Sfyrakis et al. [14] presented the VirtusCap, a capability-based access-control system that aims to facilitates the construction of least-privileged Unikernels. This tool is an access-control system integrated across the virtualization layers. The authors emphasize that their application is the first capability-based access control combined with Unikernels for the Xen hypervisor and allows Unikernel deployments to offer a potent combination of a minimal attack surface and minimum privileges, reducing the likelihoods. The evaluation also indicated that the new system makes it possible to tighten the security of Unikernels considerably, with similar performance to the XSM-Flask.
Pasquier et al. [10] proposed the PHP2Uni in order to facilitate broader adoption of Unikernels, and ease deployment of innovative solutions in resource-constrained environments. This tool performs the PHP language transcompilation to code into C++ classes. It builds Unikernels images from them as results demonstrate the transcompilation feasibility in the Unikernel context construction and preliminary comparison regarding computing and memory usage of the proposed approach against other types of deployments.
Farinier et al. [3] proposed the Jitsu, an application built to manage multi-tenant networked applications on an embedded device infrastructure securely. In order to reduce the overhead caused by the adoption and manipulation of conventional VM, Jitsu provides low latency with network services using Unikernels as the deployment unit. This tool initializes a VM in response to DNS requests in real-time. Therefore, the authors describe how to construct efficient and secure Unikernels for Xen hypervisors on ARM processors.
Madhavapeddy et al. [7] developed the Jutsu that provides a directory service that launches Unikernels in response to network traffic. This tool aims to mitigate the problems with network latency on cloud services by moving computation out to remote data centers by rapidly instantiating local services near the user. The authors indicated through results that Jitsu is a power-efficient and responsive platform for hosting cloud services in the edge network while preserving the strong isolation guarantees of a type-1 hypervisor.
This paper aims to present the modeling of Unikernels' performance and scalability during HTTP services execution. The evaluations were performed using a model to obtain the transfer rate metric among the evaluated alternatives, and the collected data was applied to a polynomial and linear progression equation. Such an approach is innovative since it was not previously performed. It is significant because it allows for understanding the performance and scalability of HTTP services when using Unikernels.

5. CONCLUSION

Traditional virtualization mechanisms provide a plethora of benefits to computing. However, in such environments, the applications present a low performance. In this sense, several studies have been developed in order to improve virtualized application performance. Currently, emerged a new technology that aims to generate a small image system to provide more security and fast booting, the Unikernel. Moreover, a Unikernel performance and scalability evaluation concerning other virtualization alternatives are still lacking. In this context, this paper aims to present a Unikernels transfer rate evaluation for HTTP services execution.
The results indicated that Rumprun Unikernels present 34,9% better performance compared to traditional VM executing over the Xen hypervisor. Each new connection, the virtual machine changes from the execution state to the paused state, waiting for a new connection. However, Unikernels have almost instant startup times,

which means it has a higher transfer rate in seconds than traditional VM over Xen. The containers still present a better transfer rate performance than Unikernels. Unfortunately, the Mirage TCP/IP library leaks memory every time a TCP connection is released, which causes the performance of Mirage to be inferior, around 30 kilobytes per second.

The scalability evaluation indicated that Unikernels have a better transfer rate with 4 instances. It occurs because the Unikernels are single-core, and the host adopted during all tests has 4 processing cores. Besides, load balancing tools generated an overhead in the transfer rate. Using only a Unikernel without a load balancing application, the results present 33% better performance than our best result, with 4 Unikernels, using these applications. It occurs because the algorithms use health checks to determine if a backend server is available to process requests. This feature aims to establish a TCP connection to the server and checks if the backend server is running on the configured IP address and port that generate overhead.

This proposal also adopted a polynomial and a linear equation to apply our obtained data during the performance evaluation. The polynomial progression was utilized to express the performance loss over the increase of Unikernels numbers. On the other hand, the Linear progression was adopted to provide the performance increase when more vCPUs are available over the same host or on the different hosts with services of load balancing and live migration that is adopted on virtualized environments.

The Unikernels are promising virtualization alternative since it provides smaller systems with only the services necessary for operation, which provides a minimum surface of attack. Besides, based on our experimental evaluation results, Unikernels are not mature enough and present more performance overheads during HTTP services execution than other well-known virtualization platforms. In this sense, the choice of virtualization technique depends on deployment requirements. If running a single shared kernel version is not an issue, containers present the fastest virtualization technique. However, with hypervisor-based virtualization, the Unikernel is a suitable choice. As future work, the authors aim to perform these tests on embedded devices and extend our experiments with OSV Unikernels.

6. ACKNOWLEDGEMENT

We gratefully acknowledge the support of National Council for Scientific and Technological Development CNPq.

7. REFERENCES

[1] I. Alobaidan, M. Mackay, and P. Tso. Build trust in the cloud computing - isolation in container based virtualisation. In *2016 9th International Conference on Developments in eSystems Engineering (DeSE)*, pages 143–148, Aug 2016.

[2] Ian Briggs, Matt Day, Yuankai Guo, Peter Marheine, and Eric Eide. A performance evaluation of unikernels. 2015.

[3] Benjamin Farinier, Thomas Gazagnaire, and Anil Madhavapeddy. Mergeable persistent data structures. In *Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA 2015)*, 2015.

[4] Antti Kantee. *The Design and Implementation of the Anykernel and Rump Kernels*. PhD thesis, doctoral dissertation. Department of Computer Science and Engineering, Aalto University, 2012.

[5] Antti Kantee. The rise and fall of the operating system, 2015.

[6] Zheng Li, Maria Kihl, Qinghua Lu, and Jens A Andersson. Performance overhead comparison between hypervisor and container based virtualization. In *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*, pages 955–962. IEEE, 2017.

[7] Anil Madhavapeddy, Thomas Leonard, Magnus Skjegstad, Thomas Gazagnaire, David Sheets, David J Scott, Richard Mortier, Amir Chaudhry, Balraj Singh, Jon Ludlam, et al. Jitsu: Just-in-time summoning of unikernels. In *NSDI*, pages 559–573, 2015.

[8] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. In *ACM SIGPLAN Notices*, volume 48, pages 461–472. ACM, 2013.

[9] R. Morabito. Virtualization on internet of things edge devices with container technologies: a performance evaluation. volume PP, pages 1–1, 2017.

[10] Thomas Pasquier, David Eyers, and Jean Bacon. Php2uni: Building unikernels using scripting language transpilation. In *Cloud Engineering (IC2E), 2017 IEEE International Conference on*, pages 197–203. IEEE, 2017.

[11] Anchal Pokharana and Rahul Hada. Performance analysis of guest vm's on xen hypervisor. In *Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on*, pages 1452–1457. IEEE, 2015.

[12] Donald E Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C Hunt. Rethinking the library os from the top down. In *ACM SIGPLAN Notices*, volume 46, pages 291–304. ACM, 2011.

[13] Amit SanWariya, Revathy Nair, and Savita Shiwani. Analyzing processing overhead of type-0 hypervisor for cloud gaming. In *Advances in Computing, Communication, & Automation (ICACCA)(Spring), International Conference on*, pages 1–5. IEEE, 2016.

[14] Ioannis Sfyrakis and Thomas Gros. Virtuscap: Capability-based access control for unikernels. In *Cloud Engineering (IC2E), 2017 IEEE International Conference on*, pages 226–237. IEEE, 2017.

[15] Bruno Xavier, Tiago Ferreto, and Luis Jersak. Time provisioning evaluation of kvm, docker and unikernels in a cloud platform. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, pages 277–280. IEEE, 2016.

[16] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A. F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '13, pages 233–240, Washington, DC, USA, 2013. IEEE Computer Society.