Performance Evaluation of a Virtualized HTTP Proxy in KVM and Docker

Rodrigo S. V. Eiras[†*], Rodrigo S. Couto[†], and Marcelo G. Rubinstein[†]
[†] Universidade do Estado do Rio de Janeiro - FEN/DETEL/PEL
[*] IesBrazil Consulting and Services - IT Section
Emails: {rodrigo.eiras}@iesbrazil.com.br, {rodrigo.couto,rubi}@uerj.br

***Abstract*—Network Function Virtualization (NFV) is a new paradigm in which network services are virtualized and can run in common and generic hardware, improving service agility and reducing costs. There are many factors that encourage NFV employment but some particular requirements must be carefully evaluated, such as choosing the virtualization technology. This paper provides a performance analysis of two open source virtualization solutions, KVM and Docker, in an HTTP proxy scenario. Our results show that Docker process HTTP requests in a lower time than KVM, due to its lightweight virtualization. Moreover, based on the results, we provide a broad discussion about each virtualization solution applied to NFV environments.**

I. Introduction

Network Function Virtualization (NFV) [1] is a recent paradigm where Network Functions (NFs), such as firewalls, proxies and NATs, often implemented as dedicated appliances, are deployed on virtual servers running on top of a common and generic hardware. Recently, the European Telecommunication Standard Institute (ETSI) started the Industry Specification for NFV [1] with the objective of standardizing the components of the NFV architecture and allowing greater accessibility to telecommunications resources for the consumer market. NFV is taking advantage from virtual environments as it allows fast scalability and on-demand implementations.

An important challenge of NFV is to provide a similar performance when compared to dedicated hardware solutions [2]. In virtualized environments, applications may have a lower performance compared to native solutions (i.e., applications without virtualization). That happens because virtual machines operations need to be translated by the hypervisor to guarantee virtual machine isolation [3]. In [4], experiments show that in a same hardware configuration, software-based routers running Linux as an operating system can achieve a forwarding packet rate of 1.2 M packets/s whereas in a Xen [5] virtualized environment the forwarding packet rate is around in 0.2 M packets/s. As a result, applications that needs a high forwarding packet rate may have a lower performance when using standard virtualization techniques, that turns into a challenge to NFV deployments. An alternative solution for VNF (Virtual Network Function) performance issues in traditional virtualized environments is the lightweight virtualization or commonly named Container-based virtualization, which isolates applications instead of an entire OS in a same operating system environment. Hence, containers are not controlled by a hypervisor, which can improve the performance, but reduces the isolation provided by the hypervisor.

978–1–5090–4671–3/16/$31.00

Given the aforementioned, an important research direction is to evaluate whether a NF can be virtualized, and which virtualization solution is more appropriated to it. In this paper, we evaluate the performance of a virtualized HyperText Transfer Protocol (HTTP) proxy. This NF, which is very important in different network types, is responsible for improving network bandwidth utilization through a cache system and filtering website accesses. According to the Squid 3 [6] documentation [7], the proxy service is very sensitive to network and CPU usage; this can turn into an issue when we virtualize this NF. Our evaluation is performed using two open source virtualization solutions: Kernel-based Virtual Machine (KVM) [8], which is a standard virtualization technique, and Docker [9], which implements lightweight virtualization through Linux containers. To accomplish this work, we used the Squid 3 proxy service as application to instantiate our VNF appliances.

The remainder of this paper is organized as follows. Section II provides an overview of related work while Section III summarizes the virtualization solutions used in our experiments. Section IV provides the experiments and Section V concludes the paper and describes future work.

II. Related Work

Several works available in the literature evaluate the performance of virtualization solutions, such as containers and virtual machines. Bondan *et al.* [10] analyze three different virtualization solutions under a network management perspective: ClickOS [11], CoreOS [12], and OSv [13], which are open source solutions considered in NFV. As they focus in network management, their metrics are related to the performance of deployment and monitoring VNFs. For example, they evaluate the time to instantiate virtual machines and containers.

Dua *et al.* [14] perform a comparison between hypervisor versus container-based virtualization technologies. They compare parameters such as security, performance, isolation, storage, and communication for each technology available to serve as Platform-as-a-Service (PaaS). They focus mainly in containers and their capacity to host applications in a large scale, but they do not evaluate performance metrics. Jing Yang and Yuqing Lan [15] propose a performance evaluation model

for virtual servers using a KVM-based virtualized system. They study factors that can impact performance on KVM-based environments and evaluate metrics such as response time for different scenarios, using one web server and a database server as services to be provided. They do not consider container-based solutions and NFV environments. The technical report [16] provides a comparison between virtual machines and containers. Different from our work, which is focused in NFV, they focus in database systems, evaluating the performance of a MySQL database.

Regarding HTTP proxies, Kim *et al.* [17] evaluate the impact of large file transfers using a web proxy cache in a high bandwidth network for a campus network environment. They measure metrics like latency, response times, and bandwidth utilization in a real environment. They do not evaluate the performance of proxies on top of any virtualized environment neither use an NFV approach, focusing in the caching system behavior across several scenarios that can cause a bottleneck.

Given the aforementioned, our contribution lies in providing a comparison between hypervisor-based and container-based virtualization, in a virtualized HTTP proxy, which is not presented in the current literature.

III. Virtualization Solutions

Virtualization solutions multiplex the access to the computer hardware allowing different virtual slices to run on top of it. In a legacy datacenter environment, without virtualization, a single computer can only run one Operating System (OS) at a time. This OS is responsible to control all the hardware devices of the computer [18], such as CPU, memory, disk controllers, disk drives, graphics cards, network cards, and many other peripherals devices. This approach is still employed in many datacenters, when only one service is offered by a given machine, generally to achieve high performance levels. However, rack space, energy consumption, scalability, and hardware upgrades [19] are some points that need attention. Virtualization can solve these challenges by allowing different OSs to share the same hardware and thus consolidating virtual slices in a single physical machine. Hence, each virtual slice has a given amount of resources from the physical server. As NFV employs the virtualization concept, VNFs can run on top of virtualization solutions originally developed for datacenters. In this paper, we classify these solutions as standard virtualization and lightweight virtualization.

A. Standard Virtualization

Standard virtualization solutions employ a software layer called hypervisor, that lies between the hardware and the virtual machines (VMs). The hypervisor controls the hardware access and performs resource allocation for each VM, providing isolation between multiple VMs. Hence, VMs employ different operating systems and host their own applications. In a nutshell, we can state that each VM has its own virtualized computer hardware environment, your own OS and thus its hardware abstraction [20]. Virtualization solutions can virtualize all common hardware devices such as CPU, memory, graphic cards, hard disk controllers, etc. Hence, there is no need to modify the VM OS to run in a virtualized environment, although some solutions modify VM OSs to improve the performance [5].

There are many hypervisors available such as VMware ESXi [21], Citrix XenServer [22], Microsoft Hyper-V [23], Oracle Virtualbox [24] and so on. In this work, we chose to use KVM [8] as the virtual machine hypervisor due to is open-source nature and the total integration with the Linux Kernel. The number of Linux servers is growing rapidly and new servers will be up to 72 million in 2017 [25]. It will be a great opportunity to use KVM as it already comes with several Linux distributions. The International Data Corporation (IDC) reports that the number of new servers deployed using KVM since 2011 is over than 278,000, representing a Compound Annual Growth Rate (CAGR) of 42% [25].

The main issues regarding the adoption of hypervisor-based approaches, such as KVM, is the performance bottleneck that it imposes in VMs. As VM operations need to pass first through the hypervisor before reaching the hardware, virtualized applications may have a lower performance compared to native solutions [4]. Hence, independent of the hypervisor employed, services implemented using virtualization techniques need to be well defined as they increase complexity and dynamics of the system, mainly when expecting NFV deployment [15].

B. Lightweight Virtualization

Lightweight virtualization, also called container-based virtualization, provides a different level of abstraction in terms of virtualization and isolation, when compared with hypervisors. As we mentioned before, hypervisors provide a hardware abstraction, which results in overhead in terms of virtualizing hardware and virtual device drivers. It means that each VM instance is a fully implemented virtual hardware to support an unmodified guest OS on top of it [26].

Different from standard virtualization, containers perform process isolation at the operating system level, avoiding overhead of the virtual hardware abstraction layer for each deployed VM. Containers run on top of the same shared operating system kernel of the physical server and one or more process can run within each container. The shared kernel provides some advantages to container based solutions, such as achieving a higher number of virtualized instances as they do not need large disk images as required for hypervisor-based solutions. As a drawback, this abstraction limits the OS distribution supported by the containers; for example, it is not possible to implement Microsoft Windows containers on top of a Linux server and vice-versa. Also, every container in a physical server runs the same OS distribution. Another common disadvantage is the isolation method provided by container-based virtualization, since the host kernel is exposed to the containers, which can be a security issue [27]. Table I performs a comparison between standard and lightweight virtualization, focusing in factors like isolation, performance, communication, and storage [14].

In our experiments, we chose Docker as container-based virtualization to serve as NFV applications. Docker is a modern open source solution to implement containers and has a large community using it in cloud environments. Docker is a daemon that provides ability to manage Linux containers as self-contained images. The main key attributes of Docker are:

- Process - Each container receives a unique Process ID (PID) and a private IP.
- Resource Isolation - Uses cgroups and namespaces concept.
- Network Isolation - Gets a private IP address bridged to a Linux interface.
- File System Isolation - Each container has its own chrooted file system.

TABLE I
COMPARISON BETWEEN STANDARD AND LIGHTWEIGHT VIRTUALIZATION

Parameter	Virtual Machines	Containers
Guest OS	Each VM runs on a virtual hardware and the kernel is loaded into its own virtual memory	All the guests share the same kernel loaded in the physical memory
Isolation	Libraries and files are completely isolated	Directories can be mounted and can be shared between the containers and the physical machine
Performance	All instructions need to be translated between VMs and the physical machine, which incurs a performance decrease	Near native performance as compared to the physical machine
Communication	Virtual Ethernet devices	IPC mechanisms such as signals, sockets, etc.
Storage	Need a large amount of disk space as each VM needs to store the whole OS and associated applications	Need lower amount of storage as containers share disks with the base OS

IV. PERFORMANCE EVALUATION

We evaluate the performance of KVM and Docker as proxy servers; i.e., using proxy as a VNF. We also evaluate a proxy in a native Linux as a baseline. Our environment includes three machines: one traffic generator, one receiver, and a proxy. The generator and the receiver are physical machines. The proxy can be a physical machine (for native Linux), a virtual machine (for KVM) or a container (for Docker).

All machines are equipped with one quad-core Xeon 3.2 GHz, 16 GB of RAM and 4 Ethernet interfaces running at 1 Gbps. To avoid external interference, the machines are interconnected using crossover cables (NIC-to-NIC). All physical machines run the same OS, Ubuntu GNU/Linux 14.04 LTS, which is also used for the virtual machine hosted in KVM.

The experiments are performed using the Apache Benchmark 2 tool [28] to generate HTTP traffic. The sender employs the benchmark to make requests to the receiver. The receiver runs an Apache 2 HTTP service, whereas a Squid 3 proxy is deployed as native Linux, container, or standard VM. All VNFs uses standard configurations with no tuning.

More specifically, our tests focus on measuring the total time needed to receive the responses for 10,000 requests. In order to evaluate the performance of the proxy in a system with high load, we insert in every test 12 fork processes on the physical server in which the proxy is executed (even when using virtualization solutions such as KVM and Docker). To perform that, we employ the Stress Tool [29] for Linux. To create the fork processes, the following command is used: *stress –cpu 12 –timeout 24h*, where the –cpu parameter is the number of fork processes to create and the –timeout parameter indicates when the stress test may terminate. Each experiment is repeated 10 times and 95-confidence intervals were plotted.

A. Processing Time Comparison with No Concurrency and No Cache System

We evaluate the performance of the whole data transfer (related to 10,000 requests) when the proxy is implemented in native Linux, KVM, or Docker. Different file sizes are requested, varying from 1 KB to 10 MB. In this first scenario, only one client sends HTTP requests and no cache system is used. Two different Docker configurations are used: routed and NAT. In Docker (Routed), traffic is routed between the physical machine and the container. In Docker (NAT), the container receives traffic directly in its socket. Due to the isolation provided by Docker, Linux uses NAT to translate the connection from the Linux socket to the Docker socket.

Table II shows the total time to reply all requests. The results show that Docker performs close to native Linux. Moreover, we can see that Docker outperforms KVM and the performance difference between them increases as the file size increases. These results can be explained since Docker shares the kernel and network drivers with the native Linux, and all the communication between the host and the containers is made through sockets. On the other hand, like any other hypervisor, KVM needs to translate all the communication between the host and the guest system, generating significant overhead in this process. When Docker is compared to the native Linux, there is also overhead in the communication because of the network isolation provided by the container virtualization system, but this overhead is much smaller than the one for KVM. In some cases, the performance is the same, considering the confidence intervals. Also, there is little performance difference between the two types of Docker configurations. Also, a CDF (Cumulative Distribution Function) is provided in Figure 1, using all samples collected in the experiments for 1MByte transfer size. As previously observed in Table II, Docker performs close to native Linux while KVM presents a large number of requests with high response time.

B. Processing Time Comparison with Concurrency and No Cache System

Next, we perform concurrent connections limited to 100 parallel users making HTTP requests. Table III show similar

TABLE II
AVERAGE OF TOTAL TIME [S] TO PROCESS 10,000 REQUESTS WITH NO CONCURRENCY AND CACHE DISABLED

Transfer Size	Linux (Native)	Docker (Routed)	Docker (NAT)	KVM
1 kB	6.98 ±0.02	7.48 ±0.02	7.48 ±0.02	9.24 ±0.27
10 kB	7.76 ±0.02	8.35 ±0.02	8.35 ±0.02	11.68 ±0.60
100 kB	15.81 ±0.02	17.28 ±0.03	17.29 ±0.03	32.53 ±1.99
1 MB	97.41 ±0.03	98.96 ±0.16	99.06 ±0.17	370.74 ±11.81
10 MB	903.63 ±0.29	913.93 ±0.80	914.76 ±0.83	2363.48 ±30.11

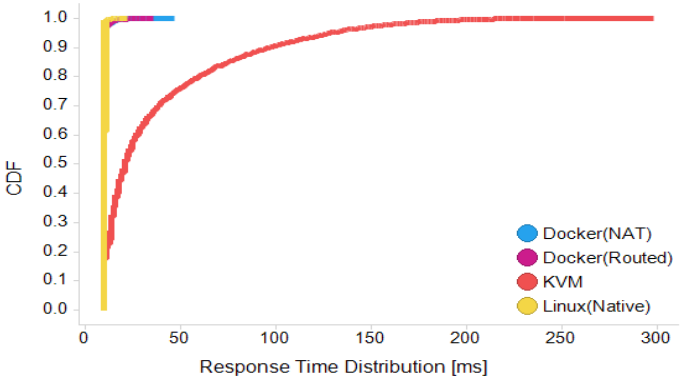

Fig. 1. Response time distribution to process 10,000 requests for 1 mByte file size with no concurrency and cache disabled.

behaviors when compared to Table II. We can note that, compared to the single user case, the total time is reduced for each transfer size since each request are performed in parallel between the 100 users. A CDF for 1MByte transfer size and 100 concurrent users is provided in Figure 2 to show how each request is performed during all the test time. Again, Docker is very close from native Linux while KVM tends to take more time to process a request, because of the overhead produced by the virtual machine layer.

TABLE III
AVERAGE OF TOTAL TIME [S] TO PROCESS 10,000 REQUESTS WITH 100 CONCURRENT CONNECTIONS AND CACHE DISABLED

Transfer Size	Linux (Native)	Docker (Routed)	Docker (NAT)	KVM
1 kB	2.05 ±0.01	2.38 ±0.01	2.49 ±0.01	3.33 ±0.02
10 kB	2.07 ±0.01	2.52 ±0.01	2.78 ±0.01	4.05 ±0.03
100 kB	8.66 ±0.05	8.53 ±0.03	8.70 ±0.04	16.99 ±0.15
1 MB	87.98 ±1.52	87.85 ±1.63	87.93 ±1.65	172.68 ±1.05
10 MB	878.06 ±20.26	877.83 ±19.06	877.75 ±19.02	1483.23 ±12.17

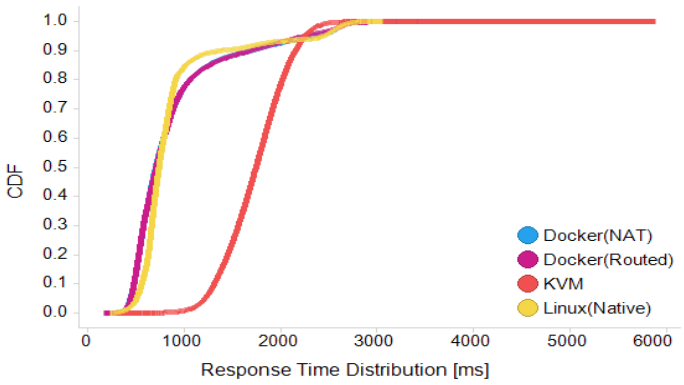

Fig. 2. Response time distribution to process 10,000 requests for 1 mByte file size with 100 concurrent connections and cache disabled.

C. Processing Time Comparison with No Concurrency and Cache System Enabled

In this section we enable the cache system on Squid 3 proxy to evaluate its performance. The cache system stores the last HTTP responses which can thus be uses by the next requests. Since this system makes an efficient use of the network, it is important to evaluate its performance and check if Docker still presents a better response time than KVM. Two parameters are set. The *maximum object size* is 1024 KBytes. The squid configuration file (squid.conf) does not encourage administrators to increase this parameter higher than this value to avoid performance issues. Because of that, when the cache system is enabled we only measure transfer sizes from 1 kByte to 1 MByte. The second parameter is the total size of the physical memory space reserved to the cache system. We employ its default value of 260 MBytes.

Table IV shows that Docker performs better than KVM even when the cache system is activated, although their performance difference is small for large file sizes. The CDF of Figure 3, for 1MByte transfer sizes, show that the performance of KVM is closer to Docker in this scenario. This happens since, as in the environment using the cache system enabled, the file is downloaded directly from the proxy server so network overhead is minimized since it is not necessary to pass the connection to the server for each request, improving the overall performance. It is important to mention that if we use containers to host proxy systems with caches enabled, like Squid 3, a large amount of memory will be consumed to store items requested from browsers. As the proxy container shares the physical memory with the host OS and may also share it with other containers, this can turn into an issue if the proxy system is not well configured.

The results using cache and concurrent connections are similar to the previous ones, and hence are not shown here.

V. CONCLUSIONS

This paper provided a performance analysis of NFV applications using Squid 3 proxy for two different types of state-of-the-art virtualization technologies, the Kernel-based Virtual Machine and the Docker container system. We evaluated the

TABLE IV
AVERAGE OF TOTAL TIME [S] TO PROCESS 10,000 REQUESTS WITH NO CONCURRENCY AND CACHE ENABLED

Transfer Size	Linux (Native)	Docker (Routed)	Docker (NAT)	KVM
1 kB	3.28 ±0.01	3.76 ±0.01	3.77 ±0.01	14.59 ±1.22
10 kB	3.84 ±0.01	4.24 ±0.01	4.21 ±0.01	17.88 ±1.38
100 kB	12.8 ±1.86	13.37 ±3.16	13.60 ±3.17	24.91 ±3.01
1 MB	92.96 ±3.12	93.54 ±3.76	93.58 ±3.86	106.68 ±3.98

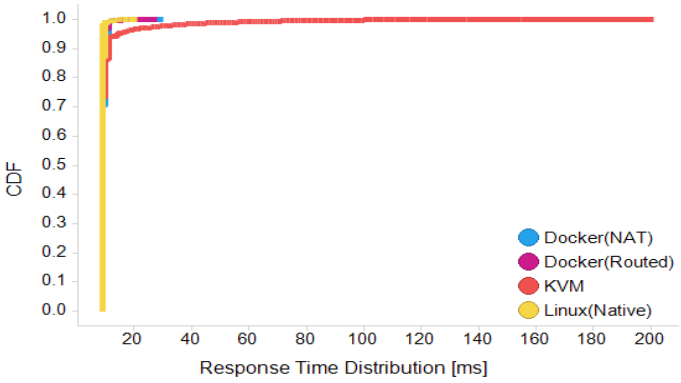

Fig. 3. Response time distribution to process 10,000 requests for 1 mByte file size with no concurrency and cache enabled.

total time needed to perform HTTP requests and receive the responses when the proxy executes in both virtualization technologies and the physical server is under high CPU load. Our main objective is to help network analysts to choose which technology is more indicated to their NFV environments.

From the experiments carried out, it is possible to conclude that Docker is more efficient to work as proxy in a NFV environment when compared to KVM-based virtual machines. As shown in the experiments, a Docker container can be two times faster than a KVM-based virtual machine, since Docker shares all the resources with the host OS and elements such as the IP table and the routing table. Also, sharing theses resources may incur in large tables in the physical server, which can be difficult to maintain depending on the environment. Hence, as a future work, we will analyze Docker scalability issues regarding table size. Also, we will perform the experiments using more than one instance for each VNF to naturally stress the physical computer and evaluate the performance in an environment closer to a real production one.

ACKNOWLEDGEMENTS

This work was funded by CNPq, CAPES, and FAPERJ.

REFERENCES

[1] N. ETSI, "Network functions virtualisation (NFV) architectural framework," *ETSI GS NFV*, vol. 2, no. 2, p. V1, 2013.

[2] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, p. 236–262, 2016.

[3] R. S. Couto, M. E. M. Campista, and L. H. M. K. Costa, "Network resource control for Xen-based virtualized software routers," *Computer Networks*, vol. 64, pp. 71–88, 2014.

[4] N. C. Fernandes, M. D. D. Moreira, I. M. Moraes, L. H. G. Ferraz, R. S. Couto, H. E. T. Carvalho, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "Virtual networks: Isolation, performance, and trends," *Annals of Telecommunications*, vol. 66, no. 5-6, p. 339–355, 2011.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM Symposium on Operating Systems Principles (SOSP 2003), pp. 164–177*, 2003.

[6] "Squid 3 proxy," http://www.squid-cache.org/Intro/ - access Aug/2016.

[7] "Squid 3 profiling and setup," http://wiki.squid-cache.org/SquidFaq/SquidProfiling - access Aug/2016.

[8] "Kernel-based virtual machine," https://openvirtualizationalliance.org/what-kvm/overview - access Aug/2016.

[9] "Docker," https://www.docker.com/what-docker - access Aug/2016.

[10] L. Bondan, C. R. P. dos Santos, and L. Z. Granville, "Comparing virtualization solutions for NFV deployment: a network management perspective," *IEEE Symposium on Computers and Communication (ISCC)*, 2016.

[11] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, , and M. Honda, "Clickos and the art of network function virtualization," *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[12] "Coreos: Open source projects for linux containers," https://coreos.com - Access Aug/2016.

[13] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, and V. Zolotarov, "OSv — optimizing the operating system for virtual machines," *USENIX Annual Technical Conference (USENIX ATC 14)*, 2014.

[14] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," *IEEE International Conference on Cloud Engineering*, 2014.

[15] J. Yang and Y. Lan, "A performance evaluation model for virtual servers in KVM-based virtualized system," *IEEE International Conference on Smart City/SocialCom/SustainCom*, 2015.

[16] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," *IBM Research Report - RC25482 (AUS1407-001)*, 2014.

[17] H.-C. Kim, D. Lee, K. Chon, B. Jang, T. Kwon, and Y. Choi, "Performance impact of large file transfer on web proxy caching: A case study in a high bandwidth campus network environment," *Journal of Communications and Networks*, vol. 52, 2010.

[18] K. Age, J. Dag, van Renesse Robbert, S. F. B., and V. S. Viken, "Omnikernel: An operating system architecture for pervasive monitoring and scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2849–2862, 2015.

[19] G. Corporation, "Energy saving via virtualization: Green it on a budget," *Gartner Tech. Rep.*, 2008.

[20] F. Yile, "Utilizing the virtualization technology in computer operating system teaching," *Eighth International Conference on Measuring Technology and Mechatronics Automation*, 2016.

[21] "Vmware esxi," http://www.vmware.com - access Aug/2016.

[22] "Citrix XenServer," https://www.citrix.com.br/products/xenserver/ - access Aug/2016.

[23] "Microsoft Hyper-V," https://msdn.microsoft.com/pt-br/library/hh831531(v=ws.11).aspx - access Aug/2016.

[24] "Oracle vm virtualbox," https://www.virtualbox.org/ - access Aug/2016.

[25] "KVM: open source virtualization for the enterprise and openstack clouds," *International Data Corporation, Tech. Rep.*, 2014.

[26] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs lightweight virtualization: a performance comparison," *IEEE International Conference on Cloud Engineering*, 2015.

[27] "Analysis of docker security," http://arxiv.org/abs/1501.02967 - access Aug/2016.

[28] "Apache benchmarking tool," http://httpd.apache.org/docs/current /programs/ab.html - access Aug/2016.

[29] "Stress tool for linux," https://people.seas.harvard.edu/ apw/stress - access Aug/2016.