2015 IEEE International Conference on Cloud Engineering

Hypervisors vs. Lightweight Virtualization: a Performance Comparison

Roberto Morabito, Jimmy Kjällman, and Miika Komu
Ericsson Research, NomadicLab
Jorvas, Finland
roberto.morabito@ericsson.com, jimmy.kjallman@ericsson.com, miika.komu@ericsson.com

***Abstract* — Virtualization of operating systems provides a common way to run different services in the cloud. Recently, the lightweight virtualization technologies claim to offer superior performance. In this paper, we present a detailed performance comparison of traditional hypervisor based virtualization and new lightweight solutions. In our measurements, we use several benchmarks tools in order to understand the strengths, weaknesses, and anomalies introduced by these different platforms in terms of processing, storage, memory and network. Our results show that containers achieve generally better performance when compared with traditional virtual machines and other recent solutions. Albeit containers offer clearly more dense deployment of virtual machines, the performance difference with other technologies is in many cases relatively small.**

Keywords — Performance; Benchmarking; Virtualization; Hypervisor; Container.

I. INTRODUCTION

Virtualization Technologies are having a very predominant role during the last years, and the number of software solutions is increasing rapidly every day. One of the reasons for adopting and deploying advanced technologies, and to build newer paradigms in this field is due, for example, to keep up with the growth of exchanged data and the consequent need to increase the capability of data center by means of server virtualization.

At the same time, the adoption of these technologies has extended to different areas, and incorporated into distinct use cases. The usage of virtualization in contexts such as *Cloud Environments, Internet of Things, and Network Function Virtualization* is becoming more widespread. The main benefits include hardware independence, isolation, secure user environments, and increased scalability, together with the large number of new properties optimized for different use cases. Consequently, the area has become very attractive and competitive, contributing to the raise of novel solutions of the main classes of virtualization technologies, that is container-based virtualization and hypervisor-based virtualization. Further, this has boosted the introduction of hybrid techniques, which promise to combine the advantages of the previous.

In this work, we show a performance analysis using different benchmark tools with hypervisor-based solutions, container and alternative solutions. The idea is to quantify the level of overhead introduced by these platforms and the existing gap compared to a non-virtualized environment.

The remainder of this paper is structured as follows: in Section II, literature review and a brief description of all the technologies and platforms evaluated is provided. The methodology used to realize our performance comparison is introduced in Section III. The benchmark results are presented in Section IV. Finally, some concluding remarks and future work are provided in Section V.

II. BACKGROUND AND RELATED WORK

In this section, we provide an overview of the different technologies included in the performance comparison. We also point out the main differences between traditional virtual machines, containers and some other emerging cloud operating systems.

A. Background

Container-based Virtualization provides a different level of abstraction in terms of virtualization and isolation when compared with hypervisors. In particular, it can be considered as a lightweight alternative to hypervisor-based virtualization. Hypervisors abstract hardware, which results in overhead in terms of virtualizing hardware and virtual device drivers. A full operating system (e.g., Linux) is typically run on top of this virtualized hardware in each virtual machine instance. In contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine, and one or more processes can be run within each container. Container-based virtualization architecture is visualized in Fig. 1(a).

Due to the shared kernel (as well as operating system libraries), an advantage of container-based solutions is that they can achieve a higher density of virtualized instances, and disk images are smaller compared to hypervisor-based solutions. The shared kernel approach has also a few disadvantages. One limitation of containers is that, e.g., Windows containers cannot be run on top of a Linux host. As another tradeoff, containers do not isolate resources as well as hypervisors [35] because the host kernel is exposed to the containers, which can be an issue for multi-tenant security.

978-1-4799-8218-9/15 $31.00 © 2015 IEEE
DOI 10.1109/IC2E.2015.74

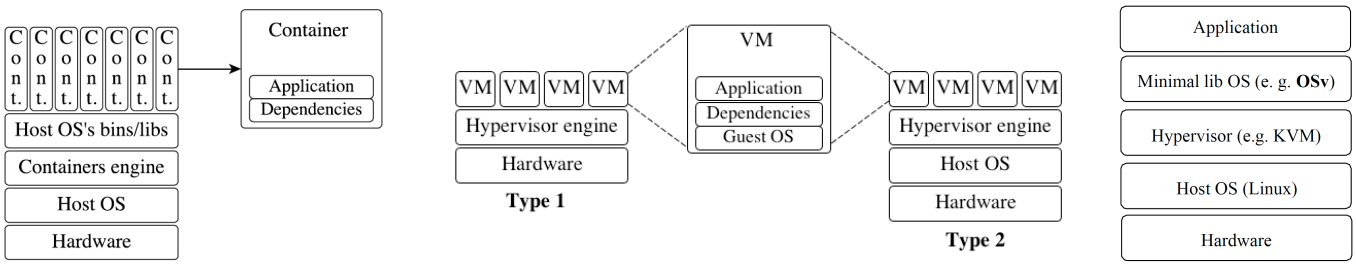

Fig. 1. Virtualization Architecture: (a) Container-based architecture [35], (b) Hypervisor-based architecture [35], (c) Library OS on a hypervisor.

In this paper, we focus on containers in Linux, which are implemented primarily via *control groups* (cgroups) [41] and *namespaces* [42] in recent Linux kernels. The former mechanism provides resource management (e.g., limits and priorities) for groups of processes, and the latter provides a private, restricted view towards certain system resources within a container (i.e., a form of sandboxing). Moreover, additional security mechanisms (e.g., SELinux) can be used for achieving more secure isolation among containers and between containers and the host system. These kernel-level features mentioned above are typically used via user-level tools.

Several container-based solutions exist, e.g., Linux-VServer [21], OpenVZ [22], LXC [23], Docker [24], and Rocket [39].

In our performance analysis, we focus on LXC and Docker. LXC is a more low-level solution that has been available for long time. Docker, on the other hand, is a higher-level platform that that has made containers very popular in a short time frame. Docker combines Linux containers with an overlay filesystem and provides tools for building and packaging applications into portable environments, i.e., container images [25].

Hypervisor-Based Virtualization has been widely used during the last decade for implementing virtualization and isolation. Contrary to containers, hypervisors operate at the hardware level, thus supporting standalone virtual machines that are independent and isolated of the host system. As the hypervisor isolates the VM from the underlying host system, it is possible to run, e.g., Windows-based VMs on top of Linux. The trade-off here is that a full operating system is installed to virtual machine, which means that the image will be substantially larger. In addition, the emulation of the virtual hardware device incurs more overhead.

According to [26], hypervisors are classified in two different types (Fig. 1(b)):

- **Type-1: native or bare-metal hypervisors** (operate on top of the host's hardware)
- **Type-2: hosted hypervisors** (operate on top of the host's operating system).

However, the distinction between the two types of hypervisors is not always so clear. For example, Linux's Kernel-based Virtual Machine (KVM) has characteristics of both types [27]. While several open source and commercial hypervisor solutions exist, we focus here on KVM [28], which is a virtualization solution for the Linux Kernel. KVM is used in conjunction with a hardware emulator and virtualizer such as QEMU.

In addition to containers and hypervisors, a number of hybrid virtualization solutions have emerged. One such solution is ZeroVM [36] that provides a single-application environment running inside NaCL sandbox from Google. Since ZeroVM operates entirely as a lightweight and restricted userspace process, it is easy to launch ZeroVM applications to process e.g. data stored on a database server. I.e., computation is brought to the data, not vice versa. A limitation of ZeroVM is that it is very challenging to port existing Linux software to it because it implements a subset of Linux functionality.

However, we focus here on another cloud OS solution called OSv [29]. It is a dedicated operating system designed exclusively for the Cloud and can be considered as a "Library Operating System" running single applications similarly as ZeroVM and Docker. In contrast to the others, OSv is intended to be run on top of a hypervisor (KVM, Xen, VirtualBox, and VMware). In other words, it achieves the isolation benefits of hypervisor-based systems, but avoids the overhead (and configuration) of a complete guest OS (Fig. 1(c)). As further benefits, OSv claims improved latency and throughput through an optimized networking stack [30], as well as a spinlock-free design with interrupt handling in ordinary threads. OSv supports Java-based applications as well as Linux-based C/C++ applications.

OSv applications can be built and executed via the Capstan tool that is conceptually similar to Docker.

OSv is a young open-source project (first Beta version has been released recently) and it has several issues especially concerning software portability/compatibility. OSv can only run executables in Linux relocatable shared object format because OSv mostly implements the ABI of Linux [34]. As OSv supports only a single process, a limited subset of POSIX, Linux syscalls, this means that Linux libc can be invoked. Other system calls, such as fork(), exec(), clone(), are not supported at the moment. Thus, porting software to OSv usually requires some effort. Some of these limitations have limited our ability to execute a deep and complete analysis, but we nevertheless provide a preliminary performance evaluation of OSv as well. Further technical details can be found in [31, 32, 33].

B. *Related Work*

Recently, the body of the literature has been focusing on optimizations to decrease the performance gap between virtualized and non-virtualized solutions. The research methodology and tools vary from study to study. The investigated and compared sets of technologies also vary from paper to paper. For example, comparisons between different hypervisors and non-virtualized systems, or even evaluation of containers vs. virtual machines have been published.

Compared to the analysis presented in this work, many of the earlier publications consider now outdated software and, perhaps even more importantly, recent work excludes new, emerging solutions. Some of such solutions, including new Library Operating System, represent a valid alternative to hypervisor and container based solutions.

Hwang et al. [1] compared four hypervisors (Hyper-V, KVM, vSphere and Xen) in different use cases. In summary, no superior hypervisor between those examined was discovered. Consequently, they propose that a cloud environment should support different software and hardware platforms to meet particular needs. A number of hypervisor comparisons have been published. Elisayed et al. [2] conduct a quantitative and qualitative evaluation of VMware ESXi5, Microsoft Hyper-V2008R2, and Citrix Xen Server 6.0.2 in various scenarios. They perform an evaluation using customized SQL instances, which simulates approximately 20 million of customers, 100,000 products, and 100,000 orders per month. Varrette et al. [3] provide a similar analysis, but with some differences. For instance, they use KVM instead of Microsoft Hyper-V2008R2, and evaluate the performance of virtualized environments for High Performance Computing (HPC) workloads in terms of power consumption, energy efficiency and scalability. The authors argue that virtualized environments for large scale HPC workloads incur a substantial performance impact by the virtualization layer (for all hypervisors), albeit some contradictory evidence on virtualization overhead does exist; Toor et al. [38] report a 4% overhead of grid virtualization. Li et al. measure a commercial (unspecified) hypervisor, Xen and KVM [4] using Hadoop and MapReduce as the use cases. Also here, the authors find similarities and significant variations in terms of performance with different workloads.

Recent research literature compares hypervisors with container solutions, including Dua et al. [5], who depict increasing use for containers in PaaS environments. Estrada et al. [6] benchmark KVM, Xen, and Linux Containers (LXC), and compare the runtime of each environment to the performance of a physical server. This lastly mentioned work is particularly interesting because of the application domain. The evaluation is based on sequence alignment software that arranges sequences of DNA. Further, Xavier et al. analyse MapReduce Clusters and in HPC Environments using containers [7, 8]. Finally, Felter et al. compare KVM and Docker performance with native environment [9]. The main difference compared to [9] is that our analysis does not consider only Hypervisor and Container solutions. In addition, the network performance section in this paper includes more results from measurements with Docker.

III. METHODOLOGY

We provide a deep evaluation with a number of benchmarking applications using KVM, LXC, Docker, and OSv. We use native (non-virtualized) performance as a base case in order to measure the overhead of the virtualization. The benchmark tools measure (using generic workloads) *CPU, Memory, Disk I/O, and Network I/O performance*.

We repeated each target of measurement with different tools to verify the consistency between the different obtained results. We also repeated each individual measurement 15 times and the illustrations show the average of such measurements. The graphs also show the standard deviation (except for some particular cases where it was insignificant).

We used the following hardware for our empirical experimentation:

- Computer model: **Dell Precision T5500**
- Processor: **Intel Xeon X5560 (8M Cache, 2.80 GHz, 4 cores, 8 threads)**
- Memory: **12 GB (3x4GB) 1333 MHz DDR3 ECC R**
- Disk: **OCZ-VERTEX 128GB**
- Network: **10Gb/s interface**
- OS: **Ubuntu 14.04 (64-bit)**

The software platforms and versions are listed in Table I.

TABLE I.

Platform	Version
KVM	QEMU emulator version 2.0.0
LXC	1.0.6
Docker	1.3.2
OSv	0.15

It should be mentioned that:

- KVM is managed using the standard Linux libvirt API and toolchain (virsh)
- OSv is running over KVM
- LXC and Docker are running directly on the host OS
- Linux Guest OS on KVM is Ubuntu 14.04 (64-bit)

From now on, we use the label "KVM" for the case where we have a Linux Guest OS running on top of the KVM hypervisor.

IV. BENCHMARK RESULTS

This section presents the results of our analysis. As explained previously, the selected benchmarks measure CPU, Memory, Disk I/O, and Network I/O performance. Results are organized in four different subsections. Section A describes all the CPU measurements. Disk I/O and Memory measurements are then shown in subsections B and C. Finally, the outcomes

of the Network performance analysis are described in subsection D.

A. CPU Performance

Y-cruncher [10] is a multi-threaded benchmark for multi-core systems to calculate the value of Pi. Y-cruncher is able to compute Pi and other constants, performing as a stress-testing application for CPU. It should be noted that other alternatives exist [11], but Y-cruncher appeared to be the only multi-threaded one. The tool provides different outputs: multi-core efficiency, computation time and total time. Total time is used to verify the results, and it consists of the total computation time plus the time required to elaborate and process the result.

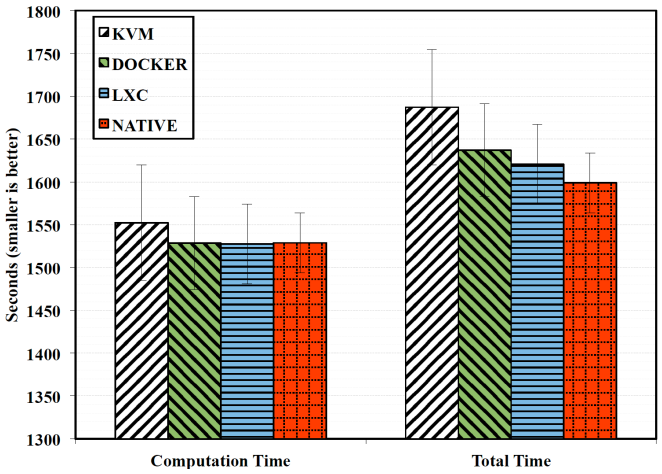

Fig. 2. Results of Y-cruncher over 30 runs. The error bars indicate the Standard Deviation. Container-based solutions perform better than KVM.

Figure 2 shows the results of the Y-cruncher tool. Both container-based solutions perform better than KVM. Considering only the computation time, containers display performance almost similar to the native environment. As shown in the next Y-cruncher measurements in Table II, even multi-core efficiency (which expresses how the CPU is efficiently used in order to execute the Pi computation) confirms this trend, even though KVM loss is relatively small (less than 1%).

TABLE II.

Platform	Multi-core Efficiency
Native	98.27%
LXC	98.19%
Docker	98.16%
KVM	97.51%

NBENCH [12] was developed during the 90s, but can still be used as a valid tool for CPU, FPU (*Floating Point Unit*), and memory system performance measurements. The tool quantifies an upper limit for the mentioned performance characteristics. Unlike Y-cruncher, NBENCH is a single-threaded tool and the algorithm performs ten different tasks producing three different indexes: *Integer Index*, *Floating Point Index*, and *Memory Index*.

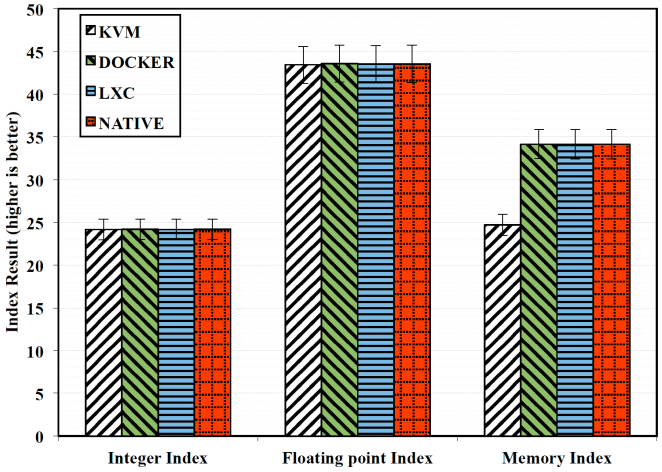

Fig. 3. NBENCH is a single-threaded benchmark tool. Except than for Memory Index (where KVM has a performance degradation compared with the other platforms), all the analysed technolgies achieve nearly equal performances.

As can be observed from Fig. 3, we can find a noticeable difference only in terms of memory index between the analysed platforms: KVM introduces roughly 30 percent performance degradation.

We confirmed the measurements of NBENCH with Geekbench [13] that is a cross-platform tool to test system stability and the processor performance. Both of the tools use the same indexing scheme for CPU efficiency, but Geekbench can index the overall system. Another difference is that it allows and supports measuring both single-core and multi-core architecture. Even considering this tool, all the compared platforms perform quite similarly. The memory index result of KVM was a bit lower, but the degradation is not remarkable as with NBENCH. Of course, the implementation of the two tests in the tools is different.

So far, the CPU tests have excluded OSv for portability reasons. However, the remaining ones will include also OSv measurements.

The "*noploop*" is a very simple CPU benchmarking tool: "*It is a procedure for measuring CPU clock speed using an unrolled No-Operation (NOP) loop. It's intended to be simple, minimizing variation caused by cache misses, stall cycles, and branch misprediction.*" [14]. Table III shows noploop results.

TABLE III.

Platform	noploop execution time (ms)
Native	2.391
LXC	2.391
Docker	2.393
KVM	2.397
OSv	2.249

We used this tool for verifying that all systems perform on the same level without major differences in this simple case. Counterintuitively, noploop performs better in OSv than in the native environment, despite the difference is minimal. Obviously, the extreme uncomplicatedness of this test does not allow us to draw accurate conclusions, since this difference between OSv and the other platforms may have been produced, e.g., by a different way of measuring time (however further analysis is needed to verify this).

Linpack benchmark exists in two variants. Intel Linpack Benchmark [16] is one, but we employed another one [15] because it was easier to port to OSv. Linpack tests the performance of a system using a simple linear algebra problem. In particular, the algorithm uses a random matrix **A** (size N), and a right hand side vector **B** that is defined as follows: **A** * **X** = **B**. The benchmark tool executes two steps in order to solve the algebra problem:

1. LU ('*Lower Upper*') factorization of **A**.
2. LU factorization is used in order to solve the linear system **A** * **X** = **B**.

Linpack gives the results in *MegaFLOPS* (millions of floating point operations per second):

$$mflops = ops / (cpu * 1000000)$$

Running the software benchmark increasing values of N, CPU behaves differently and three different "zones" can be distinguished:

- *rising zone* - local cache memory and processor are not challenged.
- *flat zone* - only the processor is challenged, that is, performing at top efficiency.
- *decaying zone* - local cache memory is challenged again, that is *"the matrix is so large that the cache is not large enough to keep the necessary data close enough to the processor to keep it running at top speed"*[15].

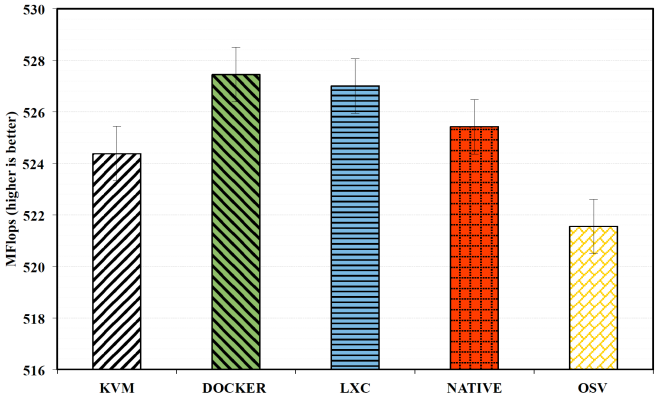

Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

Fig. 4 shows a specific use case with N=1000 that is usually used in the literature. It should be noted that the relative differences between the different platforms are not substantial. From this point of view, even if the non-virtualized environment performs worse than Docker and LXC, the gap is so small that does not allow to draw very strong implications. Fig. 5 shows Linpack results with varying N, where OSv presents some performance degradation in the rising zone, but the differences are neglectable with larger values of N.

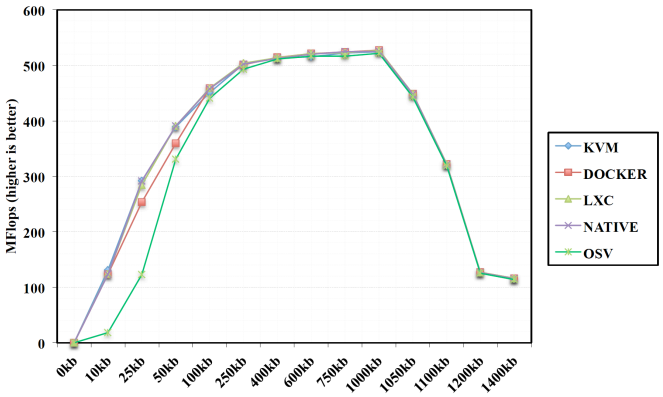

Fig. 5. Linpack results with varying N. As described in the text, three different zones can be detected: a rising zone (where OSv introduces some performance degradation), a flat zone, and a decaying zone.

B. Disk I/O Performance

Bonnie++ [17] is an open-source benchmark software that characterizes disk performance. We use it to measure disk I/O in all the evaluated systems, with the exception of OSv because Bonnie++ calls functions such as `fork()` that are not supported by OSv. It should be noted that we have configured Bonnie++ to test as recommended by the developer, that is, using a test file size of at least twice the size of system memory (in our case we consider a test file of 25GiB).

Fig. 6 shows the Bonnie++ results for sequential write (Block Output) and sequential read (Block Input) speed. The two container-based platforms offer very similar performance in both cases, which are quite close to the native one. KVM write throughput is roughly a third and read throughput almost a fifth of the native one.

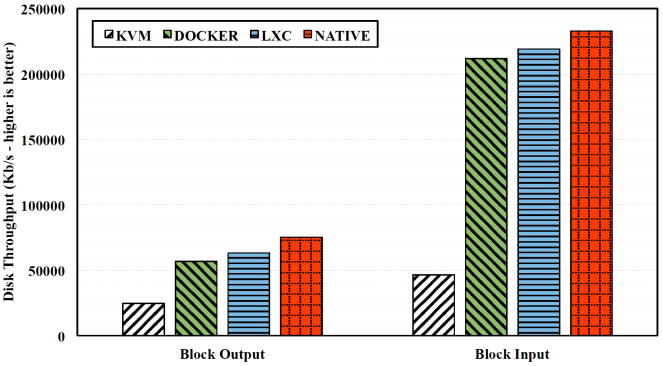

Fig. 6. Disk Throughput achieved by running Bonnie++ (test file of 25 GiB). Results for sequential writes and sequential read are shown.

Bonnie++ tool can also test the speed at which a file is read, written, and then flushed to the disk *(Random Write* test) and the number of *random seeks* per second (Table IV - for

each platform, the percentage difference respect native performance is indicated).

TABLE IV.

Platform	Random Write speed (Kb/s)		Random Seeks	
Native	48254	**%**	1706	**%**
LXC	43172	**− 10.53%**	1517	**− 11.07%**
Docker	41170	**− 14.68%**	975	**− 42.84%**
KVM	23999	**− 50.26%**	125.7	**− 92.63%**

Random Write speed results are well aligned with the results in Fig. 6. The order of the platforms is the same also in the random seek measurements, but LXC is performing now much better than Docker (approximately 30% better).

In order to get a broader understanding of Disk I/O performance, we used dd which is a simple command-line tool for Unix-like operating systems, which copies data using block size directly chosen by the user. This command can be used also for other several operations such as recovering data from hard disk, backing up function, data conversion, low-level formatting etc.

In our case, we use dd to test hypervisors and containers capacity to read and write from special device files such as /dev/zero/, that is a virtual file which has the particular feature to return null (0x00) characters when it is read. We performed this test using different block size (512 and 1024 bytes), and a test file size of 50 Gib. The following table shows the obtained results for this test (average result for block size 1024 bytes):

TABLE V.

Platform	Disk I/O speed (MB/s)	
Native	122	**%**
LXC	92	**− 24.59%**
Docker	113	**− 7.37%**
KVM	49.8	**− 59.18%**

The previous results clearly show and approximately confirm what achieved with Bonnie++. With Native, KVM, and LXC we obtained roughly always the same result without any significant deviation. Differing behavior was observed with Docker, which was performing for a few runs even better than Native (135 MB/s).

For the sake of completeness, it has to be clarified that for Native, and LXC we use the default file format for disk images. KVM uses an image format like qcow2, while Docker utilizes the AUFS *union file system*, which supports layering and enables versioning of images.

In the Disk I/O evaluation, we found a mismatch between the results of Bonnie++ and other tools such as *Sysbench* [18]. This suggests that Disk I/O performance estimation can be tricky. Also, *IOzone* [19] introduces this kind of incongruity, where LXC is more performing better than the native OS. Moreover, some of the literature [6] in the related work mentions similar unusual results. Other than this, we did not find material that compares different disk benchmarking tools for reliability, and this requires some further work.

C. *Memory Performance*

In order to test the Memory I/O, we used the *STREAM* [20] software. The extreme simplicity of this tool gave us the opportunity to perform this evaluation for all the considered platforms, including OSv. STREAM software measures memory performance using very simple vector kernels operations. It produces results for four different operations: *Copy*, *Scale*, *Add*, and *Triad*. These four operations are defined in Table VI.

TABLE VI.

Operation	Kernel
Copy	x[i] = y[i]
Scale	x[i] = q * y[i]
Add	x[i] = y[i] + z[i]
Triad	x[i] = y[i] + q * z[i]

The performance as measured by the tool has a strong dependency to the CPU cache size. For this reason, the size of the "Stream Array" in the tool has to be set properly according to the following rule: *"each array must be at least 4 times the size of the available cache memory"*. By following this condition, we obtained the results depicted in the Fig. 7. KVM, Docker and LXC all reach performance similar to the native execution. The performance of OSv is approximately half of the others according to these benchmarks.

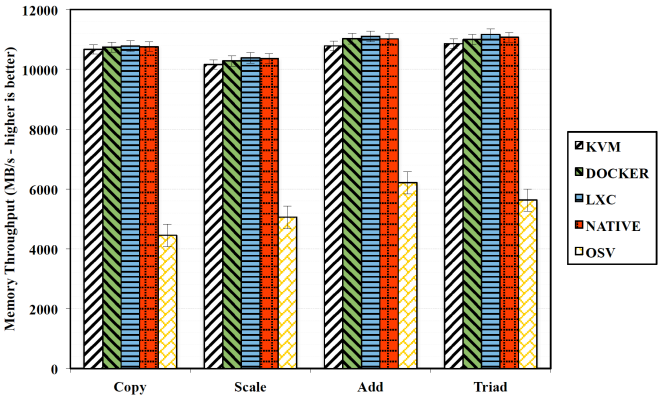

Fig. 7. Memory throughput is tested with STREAM. Measurements show that for all four operations we achieve comparable performance for each platform compared with Native distribution. OSv performs worse than the other systems in this case.

D. *Network I/O Performance*

We measured Network I/O with *Netperf* [37]. Netperf is a benchmark tool with several predefined tests to measure network performance between two hosts. More precisely, it is

possible to perform unidirectional and request/response data transfer with TCP and UDP protocol.

The configuration for the tests is as follows:

- Two identical machines directly connected with 10 Gigabit Ethernet Link.
- One host is running *netperf client* and the other *netperf server*.
- Default values for the Local/Remote socket size and the Message sizes are used.
- Test duration time: 60 seconds.
- netperf used tests: *TCP_STREAM*, *TCP_RR*, *UDP_STREAM*, *UDP_RR*.
- *IPv4* addressing
- Results represent the average across 15 runs
- netperf server is running on the tested platform

TCP_STREAM and UDP_STREAM represent the default test in netperf, that is, transmitting TCP and UDP data between the netperf client and the netperf server (netserver). Connection establishment time is not included in the measurements. The test output shows performance for processing inbound packets.

TABLE VII.

Platform	TCP_STREAM (Mbps)		UDP_STREAM (Mbps)	
Native	9413.76	**%**	6907.98	**%**
LXC	9411.01	**− 0.00029%**	3996.89	**− 42.14%**
Docker	9412	**− 0.00018%**	3939.44	**− 42.97%**
KVM	6739.01	**− 28.41%**	3153.04	**− 54.35%**
OSv	6921.97	**− 26.46%**	3668.95	**− 46.88%**

In Table VII, we can find the results for each platform and for both tests.

Using TCP, LXC and Docker achieve almost equal performance compared to the native one, and KVM is 28.41% slower. OSv performs better than KVM and it introduces a gap equal to 26.46% compared to the Native system.

All platforms offer lower throughput with UDP. LXC and Docker offer comparable performance between them (but respectively 42.14% and 42.97% lower than native); KVM overhead is the largest (54.35%). OSv ranks in the middle (46.88% worst than native). It should be noted here that the length of the transmission might have affected all the results.

The netperf request and response tests (TCP_RR and UDP_RR) evaluate the number of TCP and UDP transactions. Each transaction is determined by the following events:

- The netperf client sends request to the server
- The netserver sends a response to the client

As visualized in Table VIII, the virtualized platforms are not as reactive as the native one. For the TCP_RR test, LXC and Docker introduce a moderate level of overhead (17.35% and 19.36%) when compared to native execution. KVM offers the lowest result (47.35% slower than native). OSv performs better than KVM (roughly 4% faster).

TABLE VIII.

Platform	TCP_RR (T/s)		UDP_RR (T/s)	
Native	20742.36	**%**	21196.87	**%**
LXC	17142.67	**− 17.35%**	18901.95	**− 10.82%**
Docker	16725.26	**− 19.36%**	18623.71	**− 12.13%**
KVM	10920.48	**− 47.35%**	11495.63	**− 45.76%**
OSv	11717.3	**− 43.11%**	12050.88	**− 43.14%**

In the UDP_RR test, the relative differences between the different platforms are similar to the TCP_RR test, with. LXC and Docker that slightly reduce the performance gap (difference is now 10.82% for LXC, and 12.13% for Docker). OSv introduces roughly the same gap as for the TCP test (43.14%), while KVM is 45.76% slower than non-virtualized environment.

V. CONCLUSIONS AND FUTURE WORK

Container-based solutions and others emerging systems are challenging traditional hypervisor based virtual machines in cloud computing. The new technologies are more lightweight, thus facilitating more dense deployment of services. In this paper, we presented a detailed performance evaluation of four different technologies on Linux. More precisely, we measured Linux on KVM as an example of a hypervisor-based system running a traditional guest OS, Docker and LXC as container based solutions, and OSv as new type of lightweight guest OS.

Our results clearly show that the performance of KVM hypervisor has dramatically improved during the last few years. Nevertheless, especially Disk I/O efficiency can still represent a bottleneck for some types of applications, even though a further evaluation for Disk I/O is still needed due to some inconsistency between the different tools.

The level of overhead introduced by containers can be considered almost negligible. Taking all of the differences between LXC and Docker into account, we confirm that containers perform well, albeit the versatility and ease of management is paid in terms of security. As further work, it would be useful to repeat the measurements, e.g., with the recently announced *"Linux Container Daemon" (LXD)*[40], a new type of hypervisor that claims to offer improved support for security without losing the performance benefits. In general, security and isolation aspects of containers deserve a more thorough analysis.

OSv represents an interesting work-in-progress alternative, although it introduces some limitations in terms of software portability. Indeed, porting existing software requires some effort. However, some of the performance measurements look promising, and also the size of VM images is relatively small (in the range of hundreds of MBs).

ACKNOWLEDGMENTDS

This work is partially funded by the FP7 Marie Curie Initial Training Network (ITN) METRICS project (grant agreement No. 607728), and by Tekes with the Celtic-Plus CONVINcE Project (Project-ID CP2013/2-1). The authors would like to thank Nicklas Beijar and Tero Kauppinen of the Cloud team of NomadicLab for providing us help and support.

REFERENCES

[1] Hwang, Jinho, Sai Zeng, and Timothy Wood. "A component-based performance comparison of four hypervisors." *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013.

[2] Elsayed, Abdellatief, and Nashwa Abdelbaki. "Performance evaluation and comparison of the top market virtualization hypervisors." *Computer Engineering & Systems (ICCES), 2013 8th International Conference on*. IEEE, 2013.

[3] Varrette, Sébastien, et al. "HPC Performance and Energy-Efficiency of Xen, KVM and VMware Hypervisors." *Computer Architecture and High Performance Computing (SBAC-PAD), 2013 25th International Symposium on*. IEEE, 2013.

[4] Li, Jack, et al. "Performance Overhead Among Three Hypervisors: An Experimental Study using Hadoop Benchmarks." *Big Data (BigData Congress), 2013 IEEE International Congress on*. IEEE, 2013.

[5] Dua, Rajdeep, A. Reddy Raja, and Dharmesh Kakadia. "Virtualization vs Containerization to Support PaaS." *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014.

[6] Estrada, Zachary J., et al. "A Performance Evaluation of Sequence Alignment Software in Virtualized Environments." *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014.

[7] Xavier, Miguel Gomes, Marcelo Veiga Neves, and Cesar Augusto Fonticielha De Rose. "A Performance Comparison of Container-based Virtualization Systems for MapReduce Clusters." *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. IEEE, 2014

[8] Xavier, Miguel G., et al. "Performance evaluation of container-based virtualization for high performance computing environments." *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 2013.

[9] W. Felter, A. Ferreira, R. Rajamony, J. Rubio . An Updated Performance Comparison of Virtual Machines and Linux Containers. [Online]. Available at: http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf, last accessed 04/Dec/2014.

[10] Y-cruncher – A Multi-Threaded Pi-Program. [Online]. Available at: http://www.numberworld.org/Y-cruncher/, last accessed 01/Dec/2014.

[11] Single-Threaded Computer Benchmark | SuperPI. [Online]. Available at: http://www.superpi.net/, last accessed 27/Oct/2014.

[12] Linux/Unix nbench – Tux.org. [Online]. Available at: http://www.tux.org/~mayer/linux/bmark.html/, last accessed 03/Dec/2014.

[13] Geekbench 3 – Cross-Platform Processor Benchmark. [Online]. Available at: http://www.primatelabs.com/geekbench/, last accessed 22/Oct/2014.

[14] The noploop CPU Benchmark – Brendan Gregg's Homepage –. [Online]. Available at: http://www.brendangregg.com/blog/2014-04-26/the-noploop-cpu-benchmark.html/, last accessed 27/Nov/2014.

[15] LINPACK_BENCH – The LINPACK Benchmark. [Online]. Available at: http://people.sc.fsu.edu/~jburkardt/c_src/linpack_bench/linpack_bench.html, last accessed 23/Nov/2014.

[16] Intel® Math Kernel Library – LINPACK Download. [Online]. Available at: https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download, last accessed 23/Nov/2014.

[17] Bonnie++ now at 1.03e (last version before 2.0)!. [Online]. Available at: http://www.coker.com.au/bonnie++/, last accessed 05/Dec/2014.

[18] Sysbench in Launchpad - SysBench: a system performance benchmark. [Online]. Available at: https://launchpad.net/sysbench, last accessed 23/Nov/2014.

[19] IOzone Filesystem Benchmark. [Online]. Available at: http://www.iozone.org/, last accessed 23/Nov/2014.

[20] McCalpin, John D.: "STREAM: Sustainable Memory Bandwidth in High Performance Computers", a continually updated technical report (1991-2007), available at: "http://www.cs.virginia.edu/stream/"

[21] Linux-VServer. [Online]. Available at: http://linux-vserver.org/, last accessed 11/Dec/2014.

[22] OpenVZ Linux Containers Wiki. [Online]. Available at: http://openvz.org/Main_Page, last accessed 11/Dec/2014.

[23] Linux Containers. [Online]. Available at: http://linuxcontainers.org, last accessed 11/Dec/2014.

[24] Docker – Build, Ship, and Run Any App, Anywhere. [Online]. Available at: http://www.docker.com/, last accessed 14/Dec/2014.

[25] FAQ – Docker Documentation. [Online]. Available at: https://docs.docker.com/faq/, last accessed 14/Dec/2014.

[26] Popek, Gerald J., and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures." *Communications of the ACM* 17.7 (1974): 412-421

[27] KVM reignites Type 1 vs. Type 2 hypervisor debate. [Online]. Available at: http://searchservervirtualization.techtarget.com/news/2240034817/KVM-reignites-Type-1-vs-Type-2-hypervisor-debate, last accessed 04/Dec/2014.

[28] KVM: Main Page. [Online]. Available at: http://www.linux-kvm.org/page/Main_Page, last accessed 11/Dec/2014.

[29] OSv – the operating system designed for the cloud. [Online]. Available at: http://www.osv.io, last accessed 14/Dec/2014.

[30] Use Cases – Osv. [Online]. Available at: http://www.osv.io/user-cases, last accessed 14/Dec/2014.

[31] Kivity, Avi, et al. "OSv—Optimizing the Operating System for Virtual Machines."*2014 USENIX Annual Technical Conference (USENIX ATC 14). USENIX Association*. 2014

[32] Home • cloudius-systems/osv Wiki • GitHub. [Online]. Available at: http://github.com/cloudius-systems/osv/wiki, last accessed 14/Dec/2014.

[33] Learn More – OSv. [Online]. Available at: http://osv.io/learn-more/, last accessed 14/Dec/2014.

[34] OSv Linux ABI Compatibility • cloudius-systems/osv Wiki • GitHub [Online]. Available at: https://github.com/cloudius-systems/osv/wiki/OSv-Linux-ABI-Compatibility, last accessed 07/Dec/2014.

[35] Analysis of Docker Security. [Online]. Available at: http://arxiv.org/abs/1501.02967, last accessed 14/Jan/2015.

[36] ZeroVM sponsored by Rackspace. [Online]. Available at: http://www.zerovm.org, last accessed 01/Dec/2014.

[37] The Netperf Homepage. [Online]. Available at: http://www.netperf.org, last accessed 27/Nov/2014.

[38] Toor, S., et al. "A scalable infrastructure for the CMS data analysis based on OpenStack Cloud and Gluster file system." *Journal of Physics: Conference Series* . Vol. 513 No. 6. IOP Publishing, 2014

[39] CoreOS is building a container runtime, Rocket. [Online]. Available at: http://coreos.com/blog/rocket/, last accessed 12/Dec/2014.

[40] The next hypervisor LXD is fast, secure container management for Linux Cloud. [Online]. Available at: http://coreos.com/blog/rocket/, last accessed 12/Dec/2014.

[41] Paul Menage et al. CGROUPS. [Online]. Avaliable at: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt, last accessed 20/Jan/2015.

[42] Linux Programmer's Manual: namespaces - overview of Linux namespaces. [Online]. Available at: http://man7.org/linux/man-pages/man7/namespaces.7.html, last accessed 20/Jan/2015.