2017 IEEE 31st International Conference on Advanced Information Networking and Applications

Performance Evaluation of OS-level Virtualization Solutions for HPC Purposes on SoC-based Systems

David Beserra*†, Manuele Kirsch Pinheiro†,
Carine Souveyet†, Luiz Angelo Steffenel†‡, and Edward David Moreno*
*Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Sergipe, Aracaju, Brasil
†Centre de Recherche en Informatique, Université Paris 1 Panthéon-Sorbonne, Paris, France
‡Laboratoire CReSTIC, Université de Reims Champagne-Ardenne, Reims, France
David.Beserra@malix.univ-paris1.fr*†,{Manuele.Kirsch-Pinheiro,Carine.Souveyet}@univ-paris1.fr†,
Angelo.Steffenel@univ-reims.fr‡, edwdavid@gmail.com*

***Abstract*—Systems-on-a-chip (SoC) represents a rupture on the traditional HPC infrastructure and started to be adopted together OS-level virtualization to provide services in Fog and Edge computing scenarios. In this work, we analyzed the performance of OS-level virtualization solutions - Linux Containers (LXC) and Docker - for HPC activities running in SoC systems in order to discover how OS-level virtualization and resource sharing affects the performance of virtualized environments. We evaluated CPU and (network and inter-process) communication performance taking in account two different scenarios: without communication between benchmark processes in execution and; with processes communication in a cooperative way. Results shows that both tools are suitable for HPC applications in SoC systems, despite the notice of some performance reduction due resource sharing.**

***Index Terms*—Distributed Systems, HPC, SoC, OS-level virtualization, Performance evaluation.**

I. Introduction

High Performance Computing (HPC) is a generic term to applications that are computationally intensive or data intensive in nature [1]. While most HPC platforms rely on dedicated infrastructures such as clusters and computational grids, other technologies like cloud computing and systems-on-a-chip (SoC) are becoming interesting alternative for expensive dedicated computing infrastructures.

Indeed, cloud computing has brought a non-negligible flexibility and scalability for must users [2], and a smaller maintenance cost. One drawback, however, is that the widespread of cloud computing forced a paradigm shift as applications are no longer executed directly on baremetal but instead must be executed on top of a virtualization layer. While the performance overhead of virtualization is being rapidly reduced, it is still perceptible and may compromise some applications [3]. Another inconvenient of cloud computing is that not all applications are prone to a distant execution. Indeed, latency-sensitive applications or applications executed in remote locations with limited Internet access may be penalized by a remote execution, as well as applications relying on sensitive data that cannot be transmitted to a third-part facility.

Systems-on-a-chip (SoC), on the other hand, represent a rupture on the traditional HPC infrastructure as SoC encapsulate CPU, GPU, RAM memory and other components at the same chip [4]. Most of the times the SoC technology is used as a way to reduce the cost of single-board computers, like Raspberry PI, Intel Galileo and Banana PI. These systems are currently used for a large range of applications, from Computer Science teaching [5] to Internet of Things [6], and recent developments on SoC integration now allows the construction of computing infrastructures with a good computing power and a cost way inferior to traditional HPC platforms [7] [8]. Hence, recent works on Fog and Edge computing [9] strongly rely on SoC to bring computation closer to the user and therefore offering proximity services that otherwise would be entirely deployed on a distant infrastructure.

In this new scenario represented by HPC SoC clusters and virtualization environments, some questions arise: what is the overhead of virtualization solutions, when comparing single-board computers with traditional x86 systems? What are the main bottlenecks when deploying HPC with virtualization over SoC architectures? Hence, this work evaluates the performance of Message Passing Interface (MPI) applications and their overhead when using OS-level based virtualization solutions in SoC computers. The goal of this work is to determine what solution is more suitable for HPC in a context of resource sharing, i.e, how much the execution of containers in parallel can reduce the performance of each application. The performance measurements were conduced using traditional benchmarks of HPC community.

This work is structured as follows: Section II presents virtualization basic concepts needed to understand this work while Section III describes the related works. Section IV presents our main goals and the methodology used to perform our experiments and we present and discuss the results in Section V. Finally, Section VI presents the conclusions obtained from this study and our plans for future works.

II. Virtualization Background

Considering current virtualization technologies, we can highlight two of them: **hardware** virtualization, which makes use of Virtual Machine Monitors (VMMs), as known as hypervisors, and **operating-system-level** (OS-level) virtualization. We provide a brief description of both.

1550-445X/17 $31.00 © 2017 IEEE
DOI 10.1109/AINA.2017.73

A. Hypervisors

Hardware virtualization can be classified as Type I or Type II. Each type considers where the VMM is located in the system software stack. Type I **hypervisors** (Fig. 1.a) run directly over the host hardware and manage the guest OSs to provide resource isolation beetween VMs. In turn, the VMs can access the hardware directly or through paravirtualization.

The Type II (Fig. 1.b) virtualization relies on a VMM working inside the host OS, with the later one ensuring the access to the hardware. Type II allows creating complete abstractions and total isolation from the hardware by translating all guest OS instructions [10]. This type of virtualization is also known as **full virtualization**.

While full virtualization allows a guest OS running without kernel modifications, it imposes a higher overload in the VM performance, opposing the main goal when supporting HPC application: do not impact negatively in the performance of the system [3].

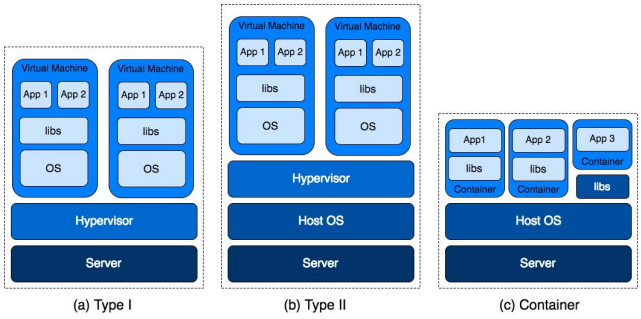


Fig. 1. Main virtualization types

B. OS-level virtualization

Besides the advantage of isolation, hypervisor-based solutions are considered heavy-weighted and have led to the development of **OS-level virtualization** (Fig. 1.c). This virtualization approach relies on OS facilities that partition the physical machine resources, creating multiple isolated user-space instances (containers) on top of a single host kernel. A container shares its kernel with the host OS. While this behavior is supposed to give a weaker isolation when compared to hypervisor-based virtualization, several works have been done to improve the isolation levels of container-based virtualization [11]. Also, there is no overhead of running other abstraction layer because OS-level virtualization does not needs a hypervisor [12]. LinuX Containers (LXC) and Docker, described below, are examples of OS-level virtualization.

1) LXC: Linux Containers (LXC) is a lightweight virtualization mechanism that does not require emulation of physical hardware. The main usage of LXC is to run a complete copy of the Linux OS in a container without the overhead of running a Type II hypervisor. On the one hand, the container shares the kernel with the host OS, thus its processes and file system are completely visible from the host. On the other hand, the container only sees its file system and process space. The LXC takes the CGroups (control group) resource management facilities as its basis and adds POSIX file capabilities to implement processes and network isolation [13]. It also uses namespaces to isolate the containers and ensure that it access only its subsets of resources. Namespaces are also used to control the network and IPC capabilities, and allow containers to be checkpointed, resumed or even migrated.

2) Docker: Docker is a high-level container-solution family that leverages LXC, differing from that by allowing the packaging of an entire environment into container images by the using an overlay file system [14]. Another relevant difference is highlighted by Morabito [15]: *"During the first releases Docker made use of LXC as execution driver, but starting with the version 0.9, Docker.io has dropped LXC as default execution environment, replacing it with their own libcontainer"*. Besides, it is possible to create personalized images that can be used as a base to the deployment of many concurrent containers. It is ideal for production environments such as commercial public clouds [16].

III. Related Works

The use of single-board computers for HPC purposes and related activities is a recent trend, being first single-board computer cluster - Iridis-PI- developed by a team from University of Southamptom in 2013 [7]. The Iridis-Pi cluster was assembled with 64 Raspberry Pi Model B acting as compute nodes and housed in a rack assembled with LEGO pieces. This cluster was designed to reach educational goals and offer a small cluster environment for tests and not to be considered as a serious alternative for HPC infrastructure. However, the authors evaluated its performance on CPU computing capacity, network throughput and latency and the capacity for I/O operations with Hadoop filesystem. The Iridis-PI delivered a computing performance near of 1.14 Gflops-1 using 64 nodes, and the subsequent analysis indicated several bottlenecks related to the low RAM memory available into nodes, and the limited network throughput (near of 100 Mb/s). The authors did not evaluate the performance of any virtulization techniques in Raspberry Pi.

Another system based in Raspberry Pi is PiCloud, a 56 Raspberry Pi cluster housed in LEGO racks [17]. Its aim was to create small cloud computing environments to emulate all layers of a Cloud stack and for use in educational activities [18]. The node interconnection was provided with an OpenFlow switch and the virtualization layer was provided with LXC, each server hosting 3 containers. No relevant study on the virtualization overhead was conducted with this environment.

A similar effort was leaded to the construction of the Bolzano Raspberry Pi Cloud [17]. This cloud environment differs from PiCloud in some aspects as it is hosted in a network infrastructure specially designed for this purpose and it associates up to 300 Raspberry Pi nodes. Also, it is not concentrated in only a single switch and does not use virtualization technology, being more similar to a conventional data center for Web services purposes than a cloud environment. Once

again, the authors do not conduct any performance analysis of the environment performance.

Some works in the subsequent years evaluated the performance of SoC for general purpose applications such as [19], who evaluated the performance of single-board computers and development boards taking in account an IoT scenario. Other works focused just on energy-consumption issues, comparing some computer-board alternatives with the use of a real Web single application [20] and with a single benchmark to measure the performance in intensive integer numbers computing [21]. While the results indicate that the ratio Performance/Watts of Raspberry Pi is far from the same ratio obtained in traditional workstations [22], the benchmarks had no specific focus on HPC applications.

Regarding HPC purposes, [23] and [24] carried out performance analysis to verify if it is possible using single-board computers for big data applications. The results pointed out the limitations due to low network throughput [24]. In [25] a ARM-based cluster performance was compared with a PC-based cluster performance, but no virtualization technology was used in the evaluation. An interesting initiative to provide a new usage of single-board computers for HPC purposes was presented in [8], which described an architecture consisting of Raspberry Pi units grouped in clusters and acessing GPUs hosted in a workstation. The goal was improve the performance of single-board computers by using virtualized slices of GPUs.

All related works presented here lack a performance analysis focusing on the virtualization support for SoC. Also, most tests were restricted to the Raspberry Pi platform, which had a limited network support (100 Mbit/s) and penalized most evaluations. Finally, no classical benchmark suites from the HPC community have been used, implying that the test did not cover critical aspects of a computer architecture, like interprocess communication performance. For all these reasons, this work proposes the analysis of virtualization environments using HPC benchmarks on the top of single-board clusters with Gigabit NICs. The experimental design proposed for this study is delineated with more details in the next section.

IV. Experimental Design

This section describes the experimental design adopted on this research. We adopted the methodology proposed in [26] as a guide to plan and execute this research. The methodology is based on four main activities: Measurement planning, Measurements, Statistical treatment and Performance analysis.

As a result, the main goal of this research is to evaluate HPC applications running in ARM-based environments and to discover which technology suits better for this architecture in a resource sharing context. To achieve the main goal, we considered the following specific goals:

1) To determine the overhead caused by virtualization on CPU-bound applications performance;
2) To determine the overhead caused by resource sharing on performance of the shared resources in function of the sharing type; and
3) To determine how virtualization affects the inter-process communication performance, considering:
   a) Communications using only intranode communication mechanisms;
   b) Communications using a physical network interface.

A. Metrics

Three main metrics have been choose for evaluating the targeted goals:

1) Amount of floating-point operations per second that could be performed by a processor in a given time (FLOPS);
2) Communication latency between MPI process pairs, measured in nanoseconds;
3) Communication bandwidth between MPI process pairs, measured in Megabits per second (Mbps).

The metrics adopted differ in function of the goal targeted. Metric 1 was used to measure the capacity of the systems evaluated in terms of processing capacity, in order to target the goal 1. The metrics 2 and 3 were used to measure the interprocess communication capacity in the systems evaluated, both using local buses, both using Ethernet network devices. The metrics 2 and 3 were used in order to target goals 2 and 3.

B. Benchmarks

To analyse the processor performance, we use the **High Performance Linpack (HPL)** benchmark. HPL measures float point operations per second (Flops) done by a computational system during a linear equations system resolution [3]. We chose HPL because it is the default benchmark used by the TOP500 ranking of the most powerful supercomputers of the world [27].

To run the HPL benchmark adequately, we need to specify the value of N, the topology of the processors grid (PxQ) and other configurable parameters. Since the N size is crucial for achieving a good performance with HPL, we use Tuning HPL[1], a Web mechanism to automatically generate the configurations to improve our experiment setup.

In addition to HPL, for understanding the impact of interprocess communication throughout, we also use the **Network Protocol Independent Performance Evaluator (NetPIPE)** benchmark. NetPIPE monitors network overheads using protocols like TCP, UDP and MPI. It performs simple ping-pong tests, sending and receiving messages of increasing size between a couple of processes, whether across a cluster connected by a Ethernet network or within a single multicore system.

C. Infrastructure

The experiments in embedded systems were conduced in a Banana Pi Model 2 (BPI-M2) single-board computer equipped with a Cortex A31S processor operating in 1 GHz, with 1 GB DDR3 SDRAM memory operating in 1600 MHz. All

[1]http://www.advancedclustering.com/act-kb/tune-hpl-dat-file/

hosts and clusters used in this experiment were configured with Debian Jessie Linux and interconnected using a Gigabit switched network. We choose Banana Pi because it has a Gigabit NIC, while Raspberry Pi has only a Fast Ethernet adapter, a strong bottleneck observed in related works.

D. Experiments performed

To reach all of the goals targeted on this research, we evaluated the environments described in Table I, taking into consideration two possible scenarios of execution:

- There is no communication between benchmark processes in execution;
- The processes communicate with each other in a cooperative way.

All environments used to perform the experiments were implemented with LXC and Docker, varying the number of virtual nodes but maintaining the same number of CPU cores and RAM. All environments used the entire available resources of a physical ARM server with 4 processor cores and 1024MB of memory. For instance, the vARM-1 environment had 1 container with 4 cores and 1024MB of available memory, the vARM-2 environment had 2 containers with 2 cores per container and 512MB of memory per container and finally the vARM-4 environment had 4 containers with 1 core per VM and 256MB of memory per container. All experiments were also performed in a bare-metal environment used as baseline, the environment Native.

TABLE I
ARM-BASED ENVIRONMENTS

Environment	Nodes	Cores/node	Memory/node (MB)
Native	1	4	1024
vARM-1	1	4	1024
vARM-2	2	2	512
vARM-4	4	1	256

All benchmarks were executed 32 times, for all tests, scenarios and conditions. In all tests, the two measurements with higher and lower value were excluded as outliers; with the other 30, we computed the average and standard deviation. Next, we describe each environment detail according to the goals aforementioned. The next subsections explain with more details the experimental procedures to reach each goal targeted.

1) Virtualization Overheads on CPU-bound Applications Performance: To evaluate the CPU performance when there is no inter-process communication, we instantiated one copy of HPL benchmark for each available processor. For instance, in the Native environment, we instantiated four HPL copies, each one executing in a distinct processor; while in the vARM-4 environment, each container had one HPL copy. Here, we observed the aggregated performance (the sum of all individual HPL copies). On the opposite, we observe explicit interprocess communications by making virtual nodes to work in a cooperative way by using a single HPL instance through all available processors. Hence, the containers in vARM-2 and vARM-4 environments were grouped in a cluster, with a single HPL copy in execution.

Each environment was benchmarked with a specific configuration description that associates the cardinality of abstractions/nodes in execution, processing resources available and HPL instances running simultaneously on the environment. The correspondence between environments and configurations adopted is shown in Table II. On this Table, 'c' means 'containers'; 'p' means 'processors'; and 'i' means 'instances'. The numbers before the letters indicates the cardinality of each element. For instance, 1c4p4i means the environment has 1 container (or physical host when in Native environment) with 4 processors available and 4 instances running simultaneously.

TABLE II
CONFIGURATIONS FOR THE ANALYSIS OF INTERPROCESS COMMUNICATION

Environment	Configuration
vARM-1	1c4p4i
vARM-2	2c2p4i
vARM-3	4c1p4i

2) Effects of Resource Sharing on Performance: In this experiment we ran a copy of HPL benchmark in a container of each type for vARM-1, vARM-2 and vARM-4 environments. The results obtained in these tests were compared with those obtained individually by each container when running concurrently with other containers of same type. In theory, equal containers have the same performances in the same activities. The correspondences between environments and configurations adopted is shown in Table III and its nomenclature are the same of the Table II.

TABLE III
CONFIGURATIONS FOR THE ANALYSIS OF RESOURCE SHARING

vARM-1	vARM-2	vARM-4
1c4p1i	1c2p2i	1c1p1i
1c4p2i	2c2p4i	2c1p4i
1c4p4i		4c1p4i

3) Virtualization Overheads on Communication Performance: In this experiment we used NetPIPE to verify the occurrence of virtualization overheads in intra-node interprocess communications. First, we ran NetPIPE in Native and vARM-1 environments; then, we ran NetPIPE in pairs of processes running in vARM-2 environment, with each process in a distinct container. In this experiment containers did not use physical network but only internal server mechanisms. This experiment is relevant since the internal communication mechanisms can vary according to virtualization approach employed, impacting the overall performance.

The physical network performance is a critical component in cluster environments. When a communication enfolds different servers or VMs hosted in different servers, the physical network is used, increasing the communication delay. To evaluate the network performance, we executed NetPIPE with processes allocated in different physical servers, measuring the performance according to packet size. To determine how the physical network virtualization affects real applications,

we also ran the HPL benchmark in virtual clusters hosted in distinct servers.

V. Results

This section presents the results obtained on this research. The results were grouped according goals and experimental method. The subsections V-A, V-B and V-C present results related to goal 1. The subsection V-D presents results related to goals 2 and 3.

A. Virtualization Overheads on CPU-bound Applications Performance

On this subsection we present the results obtained after running instances of HPL benchmark in a single server. We evaluated all configurations where it is possible to run 4 instances of HPL concurrently. Therefore, Fig. 2 shows results the aggregated CPU performance when executing HPL benchmark with no interprocess communication.

We notice that all virtualized environments have similar performances, presenting a maximum standard deviation of 3,73%. When comparing LXC and Docker, the latter presents higher standard deviations but the causes of the differences are still unclear. However some possible causes are the distinct container management library used in Docker and LXC [16] or different version of software packages installed in containers.

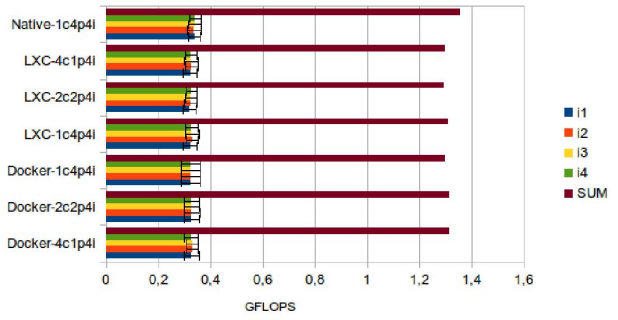


Fig. 2. HPL aggregated performance (same host)

In turn, the regularity of each individual instance is similar in both LXC and Docker, so we can conjecture that both tools presents equal share on CPU-bound applications, since these applications runs without data transfer among processes. The virtualized executions presented a performance 3,67% bellow the one in the Native environment.

Besides average and standard deviation we also computed the Coefficient of Variation (CV). It may be interpreted as data variability about the mean. Smaller values for CV indicates a more homogeneous data set and a 50% is the accepted limit value to a CV for most cases [28]. We also computed the CV for the average performance of each individual instance, obtaining a maximum CV of 11,2% (for the instance 4 running in Docker-1c4p4i configuration). This ensures the homogeneity of collected data and validates the standard deviation noticed.

Fig. 3 shows the CPU performance results when MPI processes are working in a cooperative way. We can note that execution in LXC and Docker were constant and close to Native, even when resources were shared among multiple containers. This was possible because the host system needs few resources to create and maintain a container [29], [12]. The standard deviation registered is statistically negligible, implying all environments evaluated does not presents notable oscillations on its performances. The highest sample CV was near of 3,62%, ensuring that data collected on this step are homogeneous and the standard deviations noticed are valid.

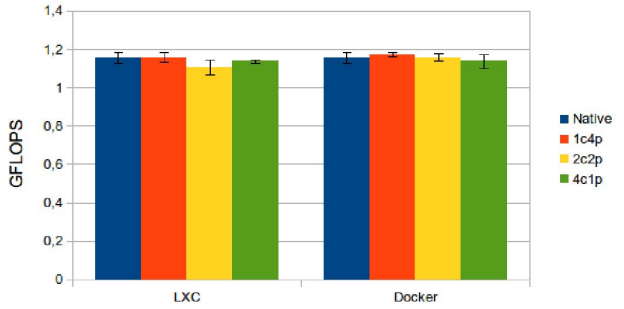


Fig. 3. HPL cooperative performance (same host)

B. Resource Sharing Overload in a Single Server

In this section, we present the results from experiments aimed to analyzing the effects of resource sharing on CPU-bound applications. For instance, Fig. 4 shows the results obtained after running a single HPL instance in containers using all hardware resources.On this context, both virtualized and native environments obtained similar performances,with a small overhead caused by virtualization but still negligible (standard deviation less than 2%).

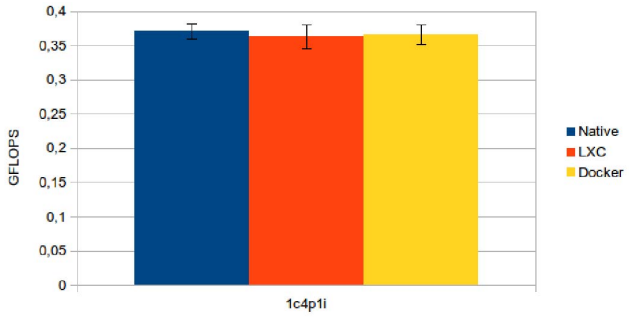


Fig. 4. Performance of 1 HPL instance running in a single container using all resources of the environment

Fig. 5 shows the results obtained after running 2 HPL instances in the different environments. The performance of each concurrent instance was compared to the performance of a single HPL instance running without competition for hardware resources. All scenarios presented a performance reduction due to resource sharing. We observed that the Native execution presented an average performance reduction of 3,94% while the virtualization scenarios presented a performance reduction of 2,97% for LXC and 3,09% for Docker. Considering the performance average and standard deviation values obtained for each instance and scenario we can conclude that resource sharing equally affected the performance of all environments.

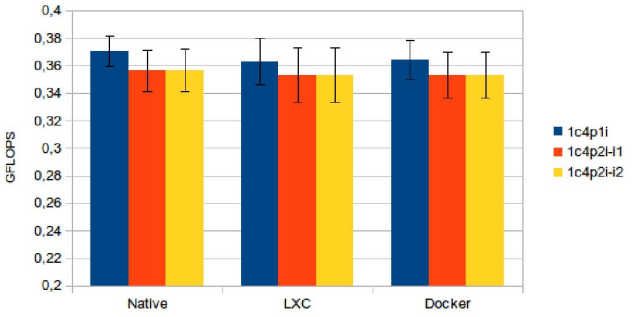


Fig. 5. Effects of resource sharing on performance with 2 instances running concurrently in a single container

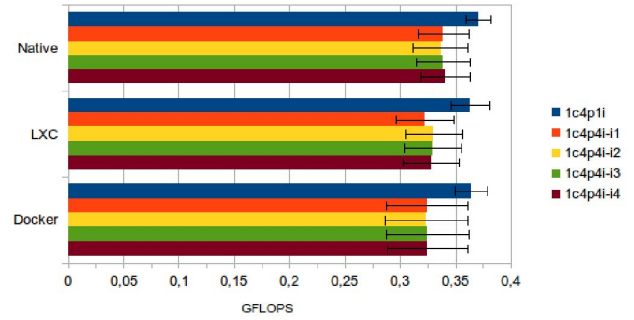


Fig. 6. Effects of resource sharing on performance with 4 instances running concurrently in a single container

Finally, Fig. 6 shows the results obtained after running 4 HPL instances and comparing them to the performance of a single HPL instance running without competition for hardware resources. The performance loss due to resource sharing grows up in all environments evaluated. Native environment shown a performance loss near 9,43% while the environment implemented with virtualization presented a performance loss close to 10,81% with LXC and 12,37% with Docker. Once again, the performance loss seem to be equally shared among all scenarios. We also observed that the more HPL instances running leads to a higher performance loss, with a reduction rate near 239% in Native, 363% in LXC and 400% in Docker when compared with the reduction rates obtained for 2 HPL instances running concurrently.

C. Resource Sharing Overload for Concurrent Containers

In this subsection we present the performance of the HPL benchmark when different instances are executed concurrently. Fig. 7 shows the benchmark results of one virtual instance running isolated against two instances of the same type running simultaneously in the same host server. Fig. 8 shows a similar comparison, but with 4 concurrent instances.

In both cases, we observe the some performance decrease in virtualized shared resources, with similar rates for each virtualization tool. When running 2 containers of the same type on vARM-2 environment, the performance of each container decreased by 9,04% when using LXC and 7,57% with Docker, in relation to same container running alone. When running 4 containers concurrrently (vARM-4 environment), the reduction

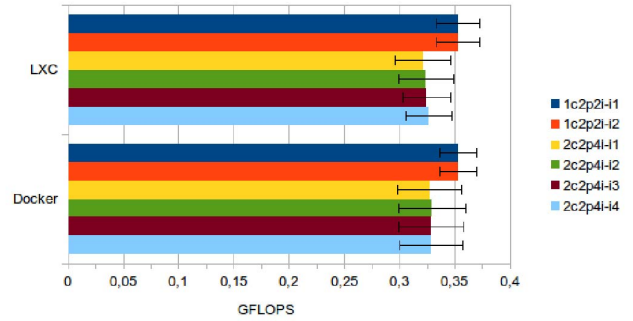


Fig. 7. Effects of resource sharing on performance with 4 instances running concurrently in 2 dual-processor containers

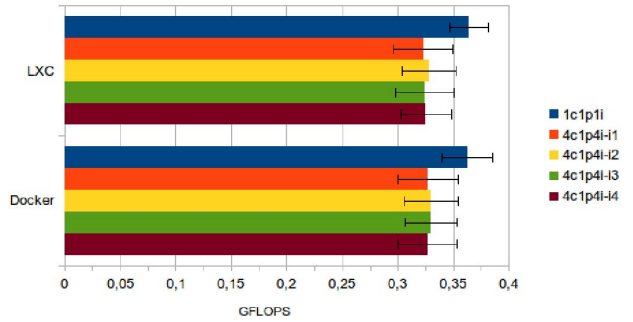


Fig. 8. Effects of resource sharing on performance with 4 instances running concurrently in 4 single-processor containers

rate jumped to 11,7% in LXC and 10,8% in Docker.

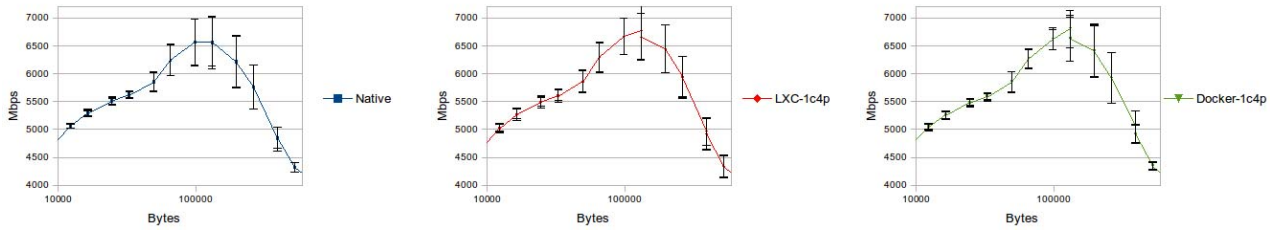
This probably occurred due to an overhead in the host server, which must provide and maintain multiple virtual resources. In this case, to avoid performance losses and a subsequent service quality below of predetermined levels in the Service Level Agreement (SLA), it is necessary to provide a flexible threshold for native host resource based in the amount of allocated containers: such SLA will allow both providers and users a stable service with respect to performance.

D. Virtualization Overheads on Communication Performance

Figure 10 shows the results of the NetPIPE benchmark in a single server. We can observe that in the 1c4p configuration on vARM-1 environment both virtualization tools achieve communication bandwidth similar to the one achieved by Native environment, and the performance variations were insignificant for most packet data sizes used.

The most significant performance variations occurred during the transmission of larger data packets between processes. Hence, Fig. 9 zooms the interval window containing the largest data packets used. The performance variations noticed on this length range may have several origins, from packet segmentation to transmission/reception queues delays.

When we use virtual environments to simulate multiple nodes, the overall communication bandwidth was slightly reduced with both Docker and LXC. Since the vARM-2 environment was configured as a cluster, the communication between NetPIPE processes exploited the full TCP/IP stack,



(a) Native (b) LXC (c) Docker

Fig. 9. Zoom on inter-process communication bandwidth results for 1c4p configuration (same host)

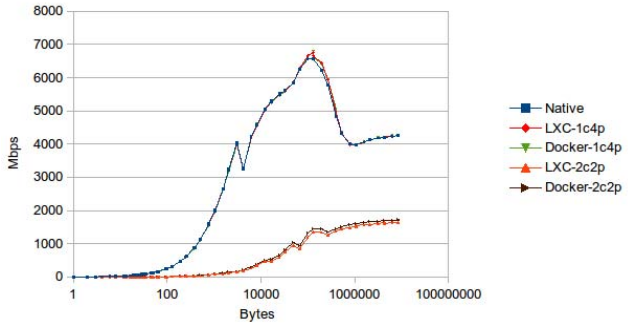


Fig. 10. Inter-process communication bandwidth (same host)

increasing the CPU usage in the host and also contributing to decrease the bandwidth. The performance differences between Docker and LXC on vARM-2 environment occurred probably due differences in the libraries used to manage the containers.

This tendency is endorsed when we observe the communication delays in Fig. 11. We found high delay values when we have more requests to process. The performance variations were insignificant for most packet sizes (less than 2%). We observed a slightly higher latency of LXC regarding Docker, which may have influenced the communication bandwidth observed in both virtualization environments.

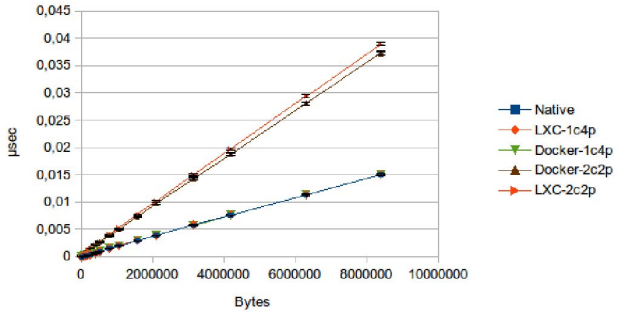


Fig. 11. Inter-process communication latency (same host)

Figures 12 and 13 show results of NetPIPE execution in two servers. All environments presented little variations for the communication bandwidthn as well as for the latency. Once more, LXC presented a performance slightly lower than Docker in terms of communication bandwidth and network latency. Nevertheless, the standard deviation lies below 5%, which is statistically insignificant.

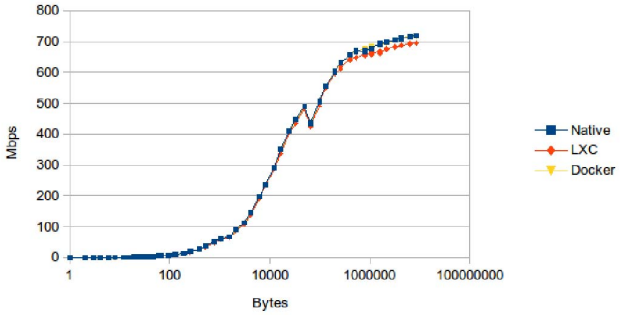


Fig. 12. Inter-process communication bandwidth (cluster)

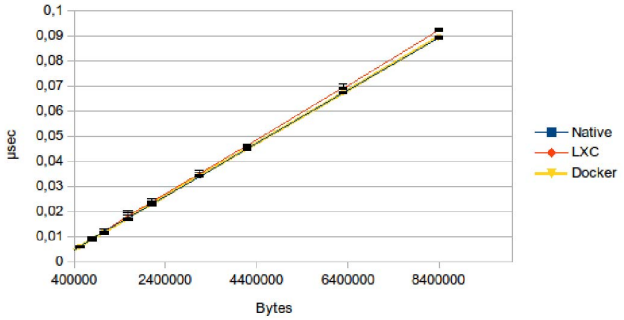


Fig. 13. Inter-process communication latency (cluster)

VI. Conclusions

This work conduced a performance evaluation between two OS-level virtualization tools: LXC and Docker. According to our experiments, we observed that both tools are suitable for HPC applications in System-on-a-Chip clusters. Considering a simple case of use, when virtual environments use all server resources for only container, both systems presented similar processing performance. In a more complex and common scenario, in which physical resources are shared among multiple and logic environments, both virtualization tools show a performance reduction, even if LXC shows a higher performance overhead. In spite of this observation, the performance reduction follows that of non-virtualized environments. Also,

all environments achieved similar performances in terms of inter-process communication capacity.

During the evaluations we were faced with some technical challenges. Indeed, the development of OS-level virtualization technologies for ARM-based systems is still a work in progress, implying a series of unforeseen events during the implementation of functional virtualized environments. Among the main challenges encountered we can cite the configuration of network interfaces on containers. This task was particularly difficult in Docker because its overlay mode, which allows hosted containers on different servers to communicate using bridges, has no support at the moment. The alternative we found was to allow direct access from the server network interface for containers in Docker. In this case, all scripts had to be run from the container and not from the host server.

Another challenge we faced was the lack of support for NFS (Network File System) on the Linux distributions currently available on ARM systems. NFS is a common filesystem on clusters dedicated to HPC, simplifying the deployment of benchmark tools and MPI code. In its absence, we had to copy the HPL benchmark executables into each container and host servers. Although this practice is still tolerated for small-scale testing, it is not suitable for large-scale systems because it allows the emergency of inconsistencies on the files and may result in management issues.

As future works, we plan to evaluate these virtualization technologies regarding the I/O performance for traditional Hard Drives (HD), as well as for SSD devices. Other aspect that can be developed in the future is the analysis of how federated and/or hybrid Clouds using ARM architectures could benefit the resource sharing when executing HPC applications.

REFERENCES

[1] T. S. Somasundaram and K. Govindarajan, "Cloudrb: A framework for scheduling and managing high-performance computing (hpc) applications in science cloud," *Future Generation Computer Systems*, vol. 34, pp. 47–65, 2014.

[2] A. Marathe, R. Harris, D. Lowenthal, B. R. de Supinski, B. Rountree, and M. Schulz, "Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2," in *23rd Int. Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2014, pp. 279–290.

[3] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," in *IEEE 4th International Conference on Cloud Computing*, ser. CLOUD '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 9–16.

[4] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 10, pp. 1701–1713, 2008.

[5] M. Ali, J. H. A. Vlaskamp, N. N. Eddin, B. Falconer, and C. Oram, "Technical development and socioeconomic implications of the raspberry pi as a learning tool in developing countries," in *Computer Science and Electronic Engineering Conf. (CEEC)*. IEEE, 2013, pp. 103–108.

[6] J. I. R. Molano, D. Betancourt, and G. Gómez, "Internet of things: A prototype architecture using a raspberry pi," in *Knowledge Management in Organizations*. Springer, 2015, pp. 618–631.

[7] S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, and N. S. O'brien, "Iridis-pi: a low-cost, compact demonstration cluster," *Cluster Computing*, vol. 17, no. 2, pp. 349–358, 2014.

[8] R. Montella, G. Giunta, and G. Laccetti, "Virtualizing high-end gpgpus on arm clusters for the next generation of high performance cloud computing," *Cluster computing*, vol. 17, no. 1, pp. 139–152, 2014.

[9] L. Steffenel and M. Kirsch-Pinheiro, "When the cloud goes pervasive: approaches for iot paas on a mobiquitous world," in *EAI International Conference on Cloud, Networking for IoT systems (CN4IoT 2015)*, Rome, Italy, Oct 2015.

[10] N. Manohar, "A survey of virtualization techniques in cloud computing," in *Int. Conf. on VLSI, Communication, Advanced Devices, Signals & Systems and Networking (VCASAN)*. Springer, 2013, pp. 461–470.

[11] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 6.

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

[13] M. Gomes Xavier, M. Veiga Neves, F. de Rose, and C. Augusto, "A performance comparison of container-based virtualization systems for mapreduce clusters," in *Parallel, Distributed and Network-Based Processing (PDP), 22nd Euromicro Int. Conf. on*. IEEE, 2014, pp. 299–306.

[14] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *Computer Engineering and Applications (ICACEA), Int. Conf. on Advances in*. IEEE, 2015, pp. 342–346.

[15] R. Morabito, "Power consumption of virtualization technologies: an empirical investigation," in *IEEE/ACM 8th Int. Conf. on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 522–527.

[16] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *Cloud Engineering (IC2E), IEEE Int. Conf. on*. IEEE, 2015, pp. 386–393.

[17] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkila, X. Wang, K. Hamily *et al.*, "Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment," in *IEEE 5th Int. Conf. on Cloud Computing Tech. and Science (CloudCom)*, vol. 2, 2013, pp. 170–175.

[18] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures," in *Distributed Computing Systems Workshops (ICDCSW), IEEE 33rd Int. Conf. on*. IEEE, 2013, pp. 108–112.

[19] C. P. Kruger and G. P. Hancke, "Benchmarking internet of things devices," in *Industrial Informatics (INDIN), 12th IEEE Int. Conf. on*. IEEE, 2014, pp. 611–616.

[20] G. Halfacre, "Raspberry pi review," 2012. [Online]. Available: http://www.bit-tech.net/hardware/pcs/2012/04/16/raspberry-pi-review/

[21] R. van der Hoeven, "Raspberry pi performance," 2013. [Online]. Available: http://freedomboxblog.nl/raspberry-pi-performance/

[22] M. Jarus, S. Varrette, A. Oleksiak, and P. Bouvry, "Performance evaluation and energy efficiency of high-density hpc platforms based on intel, amd and arm processors," in *Energy Efficiency in Large Scale Distributed Systems*. Springer, 2013, pp. 182–200.

[23] C. Kaewkasi and W. Srisuruk, "A study of big data processing constraints on a low-power hadoop cluster," in *Computer Science and Engineering Conference (ICSEC) Int. Conf. on*. IEEE, 2014, pp. 267–272.

[24] A. Anwar, K. Krish, and A. R. Butt, "On the use of microservers in supporting hadoop applications," in *Cluster Computing (CLUSTER), IEEE Int. Conf. on*. IEEE, 2014, pp. 66–74.

[25] A. Ashari and M. Riasetiawan, "High performance computing on cluster and multicore architecture," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 13, no. 4, pp. 1408–1413, 2015.

[26] E. Sousa, P. Maciel, E. Medeiros, D. Souza, F. Lins, and E. Tavares, "Evaluating eucalyptus virtual machine instance types: A study considering distinct workload demand," in *3rd Int. Conf. on Cloud Computing, GRIDs, and Virtualization*, 2012, pp. 130–135.

[27] J. Napper and P. Bientinesi, "Can cloud computing reach the top500?" in *Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop*, ser. UCHPC-MAW '09. New York, NY, USA: ACM, 2009, pp. 17–20.

[28] M. A. A. C. Guedes, T. A. and V. Janeiro, "Estatística descritiva." Universidade Federal de Maringá, 2005.

[29] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International Conference on*. IEEE, 2013, pp. 233–240.