2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

A Performance Isolation Analysis of Disk-intensive Workloads on Container-based Clouds

Miguel G. Xavier*, Israel C. De Oliveira*, Fabio D. Rossi*,
Robson D. Dos Passos*, Kassiano J. Matteussi * and Cesar A. F. De Rose*
*Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Faculty of Informatics, Porto Alegre, Brazil
Email: miguel.xavier@acad.pucrs.br

***Abstract*—The popularity of Cloud computing due to the increasing number of customers has led Cloud providers to adopt resource-sharing solutions to meet growing demand for infrastructure resources. As the adoption of resource-sharing/consolidation in Cloud computing became arguably a well-established solution, the ability the underlying virtualization systems of preventing performance interferences from customers must also be understood. Virtualization systems based on containers, such as LXC, are the basis of the next-generation of Cloud computing and have become the most popular solution under PaaS/IaaS Cloud platforms with the rise of Docker—an open platform for developers and sysadmins to build, ship, and run distributed applications. Such platforms have enticed many attentions globally, since they leverage container-based virtualization systems to offer high scalability while low performance overheads; the performance might be solely aggravated if the customers' workloads are consolidated onto the same hardware and the isolation layer does not properly isolate the shared resources. Performance isolation is an inherent concern of such systems due to the nature as they are conceived and is still an unexplored and open research topic; the consequences might influence in the adoption under shared Cloud computing platforms where Quality-of-Service is a crucial factor that cannot be disregarded. In this paper we analyze the performance interference suffered by disk-intensive workloads within very noisy-perturbed containers (different hardware components stressed). Our results show workload combinations whose performance degradation goes up to 38%, but in contrast we expose a workload-balanced scenario wherein the performance does not suffer any interference.**

Keywords*—container-based virtualization, Cloud computing, performance isolation, disk-intensive workload

I. Introduction

Cloud computing has offered a large amount of computational resources on an unprecedented scale. Users now have other alternatives for large-scale computing with more reliability, security, availability and scalability. Current public Cloud providers, such as Amazon [1] and Google [2], allow customers to allocate resources based on their needs under the pay-as-you-go model, alleviating the burdens for maintaining these resources in a private environment. This model has increased the popularity of Cloud computing and has led the providers to a scenario where the number of customers is constantly increasing, while the diversity of workloads and resource needs are steadily growing.

To deal with this abundance of customers from many fields with different resource needs, Cloud providers have used resource-sharing by leveraging virtualization to consolidate multiple customers onto the same hardware. This approach brings along significant cost savings, but also brought a level of unpredictability and an issue into focus: the performance interferences caused among customers' workloads consolidated onto the same hardware. For example, network- and CPU-intensive workloads might cause interferences among each other due to the high load placed onto the same shared network devices and processors. To address this issue, the underlying virtualization systems must be able to isolate hardware resources so that the workloads of multiple-consolidated customers do not suffer performance interferences among each other. This capability is referred to as performance isolation.

The Information Technology Laboratory (ITL) [3] has defined possible service models for Cloud computing. The Software as a Service (SaaS) model that allows customers to use provider's applications running on a Cloud infrastructure, Platform as a Service (PaaS) that provides the ability to deploy onto the Cloud infrastructure consumer-created/-acquired applications using programming languages; and Infrastructure as a Service (IaaS) that allows customers to provision different types of infrastructure resources (e.g. virtual machine, storage, network, etc) to supply their needs. Based on this, the performance isolation might be quantified/analyzed from different perspectives in Cloud computing infrastructure depending on which of these models the customers are, and which underlying virtualization system is being used. Until recently, the hypervisor-based systems were the only systems behind most of IaaS Clouds. Today, because of their slight impact on performance, container-based systems, such as LXC [4], have gained space and becoming very popular under PaaS/IaaS Clouds with the emergence of Docker [5].

Some works have emphasized the importance of having a good resource isolation layer to underpin the aforementioned models. Kreb et at. [6] propose metrics to quantify performance isolation in SaaS Clouds on which a group of customers (also known as tenants) share the same application. Other works have explored performance isolation capabilities of potential IaaS platforms, which consolidate infrastructure resources allocated by several different customers onto the same hardware [7], [8], [9], [10], [11], [12]. It is worth noting that all related works, regardless of the model, exhibit performance interferences issues in hypervisor-based Clouds, whereas understand such issues in container-based Clouds still remains an unexplored research topic, driving our motivation to conduct a set of experiments to do so. Further, our experiments consider disk-intensive workloads, as they are becoming very common in Cloud computing with the "Big

1066-6192/15 $31.00 © 2015 IEEE
DOI 10.1109/PDP.2015.67

CPS Conference Publishing Services

Data" age expansion and MapReduce applications emergence. On the same hand, Database management systems (DBMS) orchestrate large quantities of information by inputting, storing, retrieving and managing those information on disks. Such applications compute on large data sets and depends on high-throughput disk assets for sustaining their performance and satisfying customers Service-level agreement (SLA).

We focus instead on evaluating the intrinsic performance isolation issues of LXC, which is the most current container-based implementation. It has been widely adopted under the contemporary Paas/IaaS Cloud computing platforms and promise better performance and scalability compared to hypervisor-based systems, such as Xen [13] and KVM [14]. In our two earlier works [15] [16], we briefly explored performance isolation through a benchmark suite that allow us to stress CPU, memory and disk of consolidated virtual instances on HPC clusters. Evaluations of HPC workloads were conducted and the performance isolation of two types of virtualization architectures was compared. Herein, we expand and explore more in-depth performance isolation issues of a well-known virtualization system commonly used to underpin container-based Clouds. We also demonstrate the differences in regards to KVM, which is one of those used under hypervisor-based Clouds.

The rest of this paper is organized as follows: Section II briefly describes the container-based architecture and its capabilities; the contemporary container-based Clouds are also presented. Section III outlines performance isolation metrics and how they can be used to quantify virtualization systems. Section IV presents indeed the evaluations. Finally, Section VI concludes and presents future research directions.

II. BACKGROUD

A. Container-based Virtualization

Resource virtualization consists of using an intermediate software layer on top of an underlying system to provide abstractions of multiple virtual resources. In general, the virtualized resources are called virtual machines (VM) and can be seen as isolated execution contexts. There are a variety of virtualization techniques. Today, one of the most popular is the hypervisor-based virtualization, which has Xen, VMware [17] and KVM as its main representatives. The hypervisor-based virtualization, in its most common form (hosted virtualization), consists of a virtual machine monitor (VMM) on top of a host operating system (OS) that provides a full abstraction to VM. In this case, each VM has its own OS that executes completely isolated from the others. This allows, for instance, multiple different OSs on a single host.

A lightweight alternative to the hypervisors is the container-based virtualization, also known as OS-Level virtualization. This kind of virtualization partitions the physical machines' resources, creating multiple isolated user space instances on the same OS. Despite this, users in these instances have the illusion they are working on their own independent subsystem of network, memory, and file system. The main difference lies in the fact that there is no need to translate instructions from the upper to the lower layers, as such in hypervisors, wherein virtual drivers are required for this purpose. In Figure 1 is depicted the differences between the container- and hypervisor-based architectures. As can be seen, while hypervisor-based virtualization provides abstraction for full guest OSs (one per virtual machine), container-based virtualization works at the OS-level, providing abstractions directly for the guest applications. In practice, hypervisors work at the hardware abstraction level and containers at the system call/ABI layer. Since the container-based virtualization works at the OS-level, all containers share a single OS kernel. For this reason, container-based systems are supposed to have a weaker isolation compared to hypervisor-based systems. However, from the point of view of the users, each container looks and executes exactly like a stand-alone OS [18].

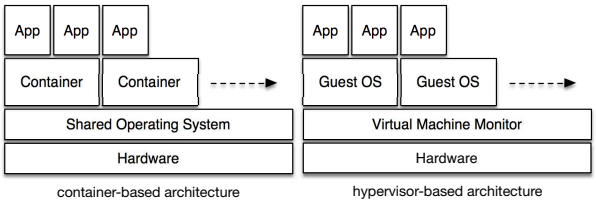

Fig. 1. Comparison of container- and hypervisor-based virtualization architectures

LXC is the today's most notable system based on containers for Linux. It has gained space because of its inclusion in Linux kernel upstream (mainline source code) and has increasingly been used in a variety of platforms that demands high scalability with low performance overheads. That is why we opted to use LXC as a representative of the container-based systems to be evaluated. The system is basically a Linux application tool that allows users to create containers containing groups of processes that might access its isolated instance of the global resource. To support container implementation, the Linux kernel has provided isolate resource capability per processes or group of processes. This feature is referred to as Kernel namespaces.

Given that containers should not be able to interact with things outside, many global resources are wrapped in a layer of namespace that provides the illusion that the container is its own system. Currently, the kernel supports six different namespaces that are responsible in turn, also by isolating different resources, which are:

- **Mount namespace:** isolates filesystem mount points seen by a container, in such way that processes in different containers might have different views of the file system hierarchy;
- **UTS namespace:** allows each container to have its own hostname and NIS domain name;
- **IPC namespace:** isolates the inter-process communication, meaning that processes containing in a containers have its own message queues, and they are completely independent from the others;
- **PID namespace:** isolates the global PID space per containers, in such a way that might have processes with the same PID number running onto different containers. It allows containers to be migrated between hosts while keeping the same applications' PID number;

- **Network namespace:** isolates the network subsystem, such as firewall tables, devices, IP address and IP route tables. Each container maintains its own networking configuration and the applications running on that can bind to the per-namespace port number space. This allows multiple web servers, for instance, to be hosted onto different containers with each server intensive to port 80 in its (per-container) network namespace;
- **User namespace:** isolates groups and users IDs from the host and other containers running on. It means that the user root (ID 0) has full privileges within a container, but without any privileges outside, ensuring safety and reliability.

Such namespaces are the basis of the LXC and in conjunction with other resource management features provide isolated user spaces environments in the form of containers. LXC takes the *cgroups* resource management facilities [19] as its basis and adds POSIX file capabilities [20] to restrict resources among the containers. By *cgroups* it is possible to apply a set of criteria to restrict resource subsystems such as memory, network, disk I/O and CPU, in the sense that one container does not exceed its imposed constraints and not interfere other containers running on the same hardware. In fact, whereas the Kernel namespaces ensure isolation among the containers such that they cannot see resources between one another, *cgroups* is responsible for resource limiting, prioritization, accounting, and control.

B. Container-based Clouds

Container-based systems have awakened attention of the industry as it brings along benefits including flexibility, agility, and scalability. Internet companies, such as Amazon and Google, have taken advantage of such systems, enabling robust container-based PaaS/IaaS platforms to build/deploy/host applications on a lightweight environment that promises to be free of performance overheads. This is the case of Docker [5]. Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications in a lightweight manner. These platforms enable apps to be quickly assembled from components, eliminating the friction between development, quality assurance, and production environments. As a result, IT can whereby ship faster and run the same application, unchanged, on laptops, data centers, and any container-supported Clouds. Also, more containers can be packed onto a single server, since the OS is not duplicated for each application. It might be a better option, if there is a need to deploy dozens or hundreds of guests. These are the main benefits that have led Cloud providers to adopt these platforms into their solutions.

On the other hand, while hypervisor-based Clouds allow customers to choose the OS that better fits their necessity, container-based Clouds are not interoperable. Linux and Windows cannot be run together [21] on the same hardware. Further, because of the shared OS, many instructions are scheduled by the same OS. This aroused Cloud providers for possible safety/stability issues, since they have no knowledge about customers, so that malicious users could affect the whole system if the isolation capabilities are not properly working.

The use of container-based virtualization under Clouds opens a new horizons for research, and all stemming questions should be understood and answered. We are taking a step, presenting in this paper a performance isolation analysis of the underlying virtualization system of the contemporary container-based Clouds.

III. Performance Isolation Overview

Uncontrolled competition for shared computing resources on a machine creates unexpected performance variations for collocated applications. In traditional shared environments, applications compete for resources. This means that an application execution might interfere the performance of another application, impacting on performance metrics. In Cloud environments, application performance should be kept because it would violate SLA, impacting directly on the user experience quality. Therefore, beyond virtualization provides other features for these environments such as elasticity and efficient use of hardware, isolation plays a decisive role. While virtual instances can share physical resources of a single machine, they remain completely isolated from each other, as if they were separated physical machines. Thus, isolation is one of the main reasons why availability and security of applications running on virtual environments are far superior to applications running on a traditional system. There is a subtle difference in container-based Clouds: there is no virtualization layer, and the OS shall ensure the performance isolation.

A simple and efficient way to implement isolation between applications or subsystems in container-based virtualization lies in the user space. In this approach, the OS's user space is divided into areas referred to as isolated virtual domain. Each virtual domain allocates an OS resources portion, such as memory, CPU time and disk. In addition, some features of the real system can be virtualized, like network interfaces– each domain has its own virtual interface and its own network address. This notion of distinct spaces can be extended to other system resources as we presented in section II-A.

Although the customers have no control over the Cloud infrastructure, they need assurances of reliability and performance of the contracted resources, since the migration involves transferring vital functions from their business to the Cloud (i.e. it becomes vital for customers to obtain guarantees of delivery service providers). Typically, these guarantees are provided by SLAs negotiated between providers and customers, but this is not a straightforward task, especially in the Cloud computing infrastructures. The simplest way is through committed resources dedicated to each customer. Many Cloud providers strive to cut infrastructure costs through excessive subscriptions in their data centers, so that the resource capacity became underutilized and the isolation capability becomes an even greater concern [22].

There have been works proposing the use of metrics for quantifying performance isolation between virtual domains, such as response time and disruptive loads, on several scenarios (non-isolation virtual domains against pinned and unpinned virtual domains) [6] [8] [9] [23]. The findings have revealed that the metrics are enough to prove that the system is completely isolated, however they fail at ranking the isolation system resources into the range between isolated and non-isolated.

In a more restrict view, the most usual way to quantify performance isolation of workloads in Cloud environments,

consists in running concurrent workloads within different virtual domains, whose workloads consume equally the same physical resource, as can be seen in [24] [25] [26] [27] [28]. In Figure 2 is illustrated this scenario applied in the context of container-based virtualization under disk-intensive workloads.

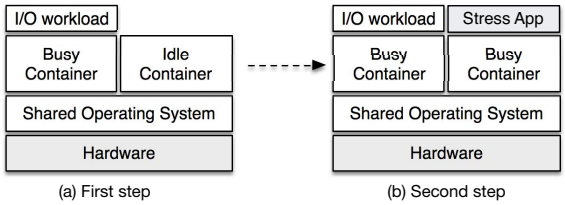

Fig. 2. Isolation Evaluation Architecture

As observed, the total amount of physical resources is partitioned evenly among the containers (two containers per machine). The workflow consists basically of two steps. In the first step (a), a disk-intensive workload runs alone into one of the two containers, while the other remains idle, meaning that no container has been disturbed. The disk-intensive workload metric is collected and stored to be used further, at the end of the workflow. In the second step (b), the same workload runs into one of the containers, but now together with a disruptive container running side-by-side. The metric is collected again and the performance isolation is quantified by the difference of the metrics collected from the two steps, denoted as follows:

$$D = \left[1 - \left[\frac{Domain_{disruptive}}{Domain_{base}}\right]\right] * 100 \quad (1)$$

The difference D denotes the performance degradation and might give an insight whether the namespaces and *cgroups* are properly isolating the resources. In fact, forecasting performance isolation of virtualized applications is quite beneficial for large complex data centers, not only to improve the relatively static allocation of workloads to physical resources, but also as input to more dynamic orchestration systems.

IV. Evaluation Results

Experiments were conducted to evaluate the performance interference a disk-intensive workload suffers by disruptive virtual domains running on the same hardware. To that end, we chose Oracle [29] and MySQL [30] databases for the sole purpose of putting load stress on them and contrasting an open-source and a commercial DBMS. Regarding Oracle, it is used in environments that require high scalability and robustness. MySQL, however, is broadly used in Internet applications such as e-commerce sites. Both systems have been supported by Oracle, but Oracle databases are commonly indicated for business enterprises, while MySQL is a lighter and straightforward solution purpose-built for small companies that dispense high investments in supporting and maintaining. We used respectively the versions 12c and 5.5 of Oracle and MySQL.

To put stress on these systems, we ran the Swingbench [31] and the Sysbench [32] benchmarks on Oracle and MySQL databases, respectively. They are both TPC-C benchmarks [33] and are essentially based on On-Line Transaction Processing (OLTP) classes that facilitate and manage transaction-oriented applications. They enable multiple concurrent users' transactions (e.g. select, update and delete operations) to be executed on the DBMS systems. In fact, OLTP-oriented benchmarks emulate typical transactions on systems that need high availability, while dealing with several simultaneous users' requests. There are five basic transactions that represent the behavior of an OLTP system. These transactions and their characteristics are outlined in Table I. The transaction percentage is analogous to the distribution used by an OLTP system [33].

TABLE I. TPC-C Transactions types and occurrences

Transaction	Characteristics	Percentage
New Order	read-write, middle complexity	45%
Payment	read-write, low complexity	43%
Order Status	read, middle complexity	4%
Stock Level	read-write	4%
Delivery	read-write	4%

By the nature of the DBMS systems, they can sustain finite floods of transactions over a certain period. Such floods refer to the number of atomic actions performed by certain entity per second. This measurement has become a universal metric used in database warehouse for measuring performance in terms of database processing capacity, since it can give a vision of how efficient the system is (how many transactions the system is capable of executing without impact on user satisfaction). This metric is referred to as Transaction Per Second (TPS) and it is the output of popular TPC-C benchmarks such as those used in our evaluations, which are: Swingbench and Sysbench. TPS refers to the disk-intensive metric collected from both steps in the workflow earlier described in section III.

Rounds of tests were carried on the workflow and the number of rounds was given by a confidence interval of at least 95%. The metric TPS obtained from the first step was simply collected via the benchmarks and stored to be used as a factor of comparison further. In the second step, while the DBMS system within a domain was being stressed via a benchmark outside, the other domain began having its allocated hardware resources stressed (CPU, disk, and memory) through purpose-build apps. The apps aim to put stress on the domains in an attempt to cause performance interferences to the domains that are handling database transactions. The stress app called *memorybomb*, basically loops many memory allocation instructions in order to reach the limits imposed by the OS. Similarly, the *cpubomb* stress app loops arithmetic operations, putting stress on all CPU cores simultaneously. To stress the disk I/O, we used the IOzone filesystem benchmark [34] that allowed us to measure read/write file operations fairly accurately.

A. Experimental Setup

Our hardware setup comprises two identical Dell PowerEdge R810 machines. Each of them equipped with two 3.46Ghz Intel Xeon C5690 processors with 8 cores each (with Hyper-Threading), totaling 32 virtual cores; 64Gb of RAM memory, and four Gigabit Ethernet adapters. The communication between them is done via a Gigabit switch. On these machines we deployed the Linux distribution Ubuntu 14.04 LTS [35] and the OpenStack Cloud platform (Havanna Release) [36]. In summary, our testbed is an OpenStack environment composed of two nodes, where the system that orchestrates

the virtual domains in such nodes are eventually swapped depending on which is being evaluated: the container-based LXC or KVM, which is the hypervisor-based representative to be collated.

To perform the experiments on LXC, we installed the LXC toolkit (1.0.5) on it and ensure that all requirements presented by the *lxc-checkcong* tool were met. Finally, each container allocates half of the total amount of resources of CPU, Disk I/O and memory, taking into account that the workflow cycles consider only two containers in place. The *cgroups* was used for this purpose.

Experiments involving KVM were performed with the purpose of comparing the results with the ones obtained from LXC. We installed on our testbed the KVM (version 1.0) and the QEMU (version 1.4) [37] to support all virtualization capabilities and to conduct the experiment in an environment as similar as possible with the LXC.

B. Performance Isolation Analysis of LXC

In this section we discuss the outcomes of performance interferences caused on the DBMS systems stemming from a disruptive LXC container running on the same OpenStack node. On the experimental platform, we conducted experiments using Oracle and MySQL to ascertain any perturbations in their operations/performances while handling database transaction-oriented floods carried from the benchmarks.

Figure 3 depicted the TPS metric obtained upon the MySQL. Albeit there seems to be almost no disturbance throughout the execution of all stress apps, it does not mean that LXC subsystems are playing their role and all resources are tightly isolated. Due to the Sysbench limitations and because of its characteristic, we were not able to separately benchmark different TPC-C transactions and infer the one that suffers more performance interferences. This led us to diversify and contrast the results with other DBMS system like Oracle.

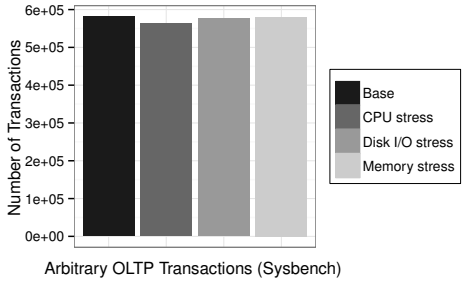

Fig. 3. Performance Isolation on LXC - MySQL

While observing the Oracle results in Figure 4, the differences become more noticeable. The Y-axis scale (number of transactions) differences in due to the Swingbench limited execution time. The execution time impact directly on the number of transactions the benchmark is able to supply, but it does not affect the analysis among the TPC-C transactions executed by the Swingbench. Careful examination of the outcomes, reveal that all TPC-C transactions suffer some impact caused by the stress apps, being the *memorybomb* the most noisy one. This led us to believe that the *cgroup* memory restriction capability was not working well and its behavior merited to be further studied.

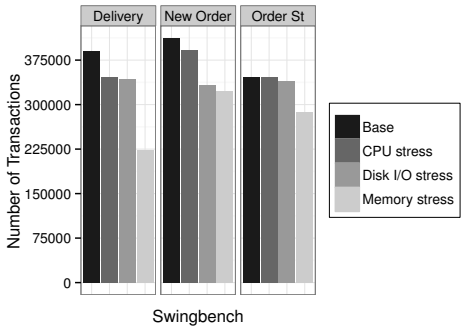

Fig. 4. Performance interferences in the database transactions on LXC with Cgroups

Going further, we explored this troublesome issue regarding memory constraint in *cgroups* by carrying out a new test set. We found that there seems to be a relationship with the Out-of-Memory (OOM) killer of the OS. The Figure 5 depict out our assumption. As noted in the labeled time-slice A, at the time the *memorybomb* app is killed by the Linux kernel to conserve its stability and avoid crashing, the performance of the database goes down and take some time to go back on. Throughout this time, the number of transactions keeps at 0. This intermittent behavior reflects on the totality of transactions reached by the Swingbench along its execution, as perceived in Figure 4.

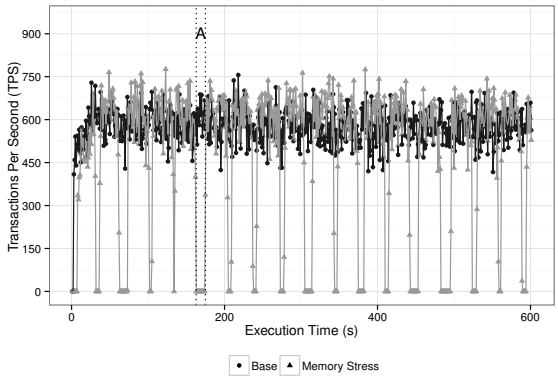

Fig. 5. Performance impacts suffered by the memory stress app. The performance results from the workload within the perturbed LXC container overlapped onto the results from the LXC container without any induced interference (i.e. steps (1) and (2) are contrasted)

C. Performance Isolation Analysis of KVM

There are two ways to restrict the resources in KVM: One is by using the built-in QEMU features and the other is by using *cgroups* like in LXC. Our tests covered both in order to identify which of them contributes to better resource constraints and provides a fairer comparison with LXC.

As noted in Figure 6, in spite of not having driven its performance by the stress apps, the TPS reached herein is

much lower than the TPS obtained from LXC, as we saw in Figure 4. On the other hand, the performance isolation ability of KVM is far superior, ensuring more security/reliability with almost no interference. Such differences are explained by the intrinsic virtualization layer of the hypervisor-based systems, and the absence of such layer in container-based systems. These benefits/drawbacks is better detailed in our earlier work, since it is not in the ongoing work context.

Fig. 6. Performance interferences in the database transactions on KVM with Cgroups

Fig. 7. Performance interferences in the database transactions on KVM with QEMU

It is easily observing in Figure 6 and 8 that all tests, regardless of the database and the type of TPC-C transaction carried out, did not suffer much interferences while all resources were stressed. This suggests that the *cgroups* is working properly when using it to restrict resources in KVM. In contrast, the own QEMU resource management did not express the same behavior, as can be seen in Figure 7.

For better interpretation, we summarized the performance degradations in Tables II and III in which both the Oracle and MySQL results are exposed. When comparing the results from LXC against the results obtained from KVM in which the *cgroups* is exercised, we can infer that all resources limited by *cgroups* in both environments did not interfere the performance of the databases. This suggests that interferences in LXC are due to the shared OS that had to handle much more instruction during high load conditions, being unable to efficiently handle the database transactions.

We realized considerable performance interferences while the memory limits were stressed. The most notable was during

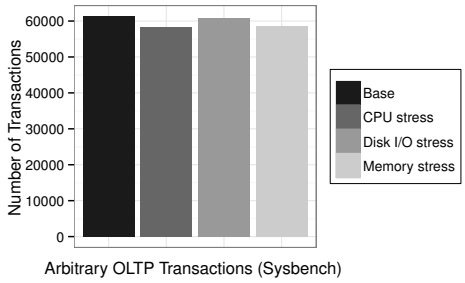

Fig. 8. Performance Isolation on KVM - MySQL

TABLE II. PERFORMANCE DEGRADATION OF SWINGBENCH (ORACLE) FOR ALL STRESSED RESOURCES

Operations	D_{CPU}	D_{disk}	D_{memory}
LXC - New Order	5,04%	19,21%	21,62%
LXC - Delivery	11,37%	12,35%	38,90%
LXC - Order St	0%	1,87%	16,74%
KVM (Cgroup) - New Order	0%	3,43%	4,16%
KVM (Cgroup) - Delivery	4,65%	4,17%	5,46%
KVM (Cgroup) - Order St	5,00%	3,71%	4,60%
KVM (QEMU) - New Order	8,49%	26,49%	19,89%
KVM (QEMU) - Delivery	7,93%	5,91%	3,85%
KVM (QEMU) - Order St	0,28%	4,12%	3,08%

TABLE III. PERFORMANCE DEGRADATION OF SYSBENCH (MYSQL) FOR ALL STRESSED RESOURCES

Operations	D_{CPU}	D_{disk}	D_{memory}
LXC - MySQL OLTP	3,22%	0,88%	0,10%
KVM (QEMU) - MySQL OLTP	27,14%	21,85%	8,88%
KVM (Cgroup) - MySQL OLTP	4,89%	0,65%	4,65%

the *delivery* transactions, since this kind of transaction takes many memory pages to store and handle large data sets before writing them back to the database. A similar behavior was also observed in [15].

As noted, the container-based system is not yet mature to ensure performance isolation among disk-intensive workloads. However, consolidate different workload types is a good alternative to alleviate interferences. Based on our results, we would suggest not combining disk- and memory-intensive workloads into different containers due to the inherent performance degradation observed while putting them together onto the same physical machine. On the other hand, we observed that by consolidating I/O- and CPU-intensive workloads, it is possible to alleviate the performance impacts.

V. RELATED WORK

Virtualization has brought a great flexibility to the computing infrastructures. Although all the known advantages, tradeoffs between I/O isolation [38] and performance [39] are still issues that are widely discussed in the literature. In this way, Soltesz et al. [40] evaluates the impact of VServer performance against a traditional Xen virtualization environment, using some performance metrics such as bandwidth, CPU and memory usage, and disk access. The work concluded that the

container-based environments can be up to two times faster than traditional virtualization environments.

The paper by Pu et al. [41] evaluates the interference of I/O applications among multiple virtual machines. The SPEC benchmark was used on two Xen virtual machines sharing the same resources, and monitoring CPU and network usage. The results showed that the performance of I/O workloads on isolated environments shows a high overhead due to the constant changes in context between the host system and virtual machines. Also, isolation is not as efficient because there is a lack of contention due to demand for faster memory pages in I/O channel exchanges. Other questions are presented, related to the interference of CPU-intensive workloads and network-intensive workloads on the same physical machine can bring overhead due to competition for resources.

The work of Mandal et al. [42] evaluates the I/O network on Cloud environments, and what impact the isolation of virtual machines has on this issue. Several provisioning scenarios were tested using HTCondor, and the results showed that the allocation of the bandwidth increases to the point where the disk I/O is saturated. This enabled the identification of a maximum useful point in the allocation of network bandwidth. Thereafter, there is little or no benefit for performance.

The paper by Mei et al. [43] presented a performance evaluation based on network I/O applications in virtualized datacenter environment. To quantify the impact of these applications on the environment, the test scenario consisted of the httperf benchmark performing network requests to web servers on distributed virtual machines. Several scenarios for the use of virtual machines were tested, since comparison of virtual machines with load against idle virtual machines running on the same physical machine, until scenarios with all virtual machines at full load. The results showed that this type of I/O application is affected due to poor isolation of memory.

Fang et al. [7] proposed a mechanism for allocating I/O applications with the aim of reducing the impact of the weak isolation of virtual machines, in a way that the mechanism can recognize the behavior of the application and adjust the workload parameters, eg, amount of reading and writing operations. The initial tests were performed on Xen and VMWare, and then extended to the Amazon EC2 platform. After the profiling phase, a mathematical model was created trying to predict the behavior of I/O applications. The results showed a standard deviation of 1,04%.

As we can see, traditional virtualization environments are being widely evaluated aiming to reduce the trade-off between performance and resource use. New virtualization proposals are continuously emerging, such as the container-based, which while allowing higher performance than traditional virtualization environments, it still leaves something to be desired in the isolation support. Nevertheless, it is a quoted technology to support future Cloud environments [44].

The works proposed by Xavier et al. [15] [16] analyze the isolation of container-based environments with different workloads. The first paper [15] presented an evaluation of several container-based environments, such as LXC, VServer and OpenVZ, using as a basis of comparison a native Linux environment, and a traditional virtualization environment such as Xen. The tests evaluate several performance metrics using several benchmarks: Linpack was used to assess the performance of computing, STREAM to memory performance, IOzone to disk performance, NetPipe to network performance, beyond NAS Parallel Benchmark to assess HPC applications. The results showed that container-based technology is not yet mature in isolation of performance issues because of a container application can interfere with the performance of other containers. The second paper [16] evaluates the performance of MapReduce applications on container-based environments. In the work, performance evaluation of the file system (HDFS) was performed. The results showed a better isolation of LXC when compared with other container-based platforms proposed, makes its use very attractive for MapReduce applications.

This paper proposed an extension of these latter two papers, assessing disk-intensive workloads on container-based Clouds. Considering the weak support issues of isolation in container-based systems, disk-intensive workload can impact in a very decisively in these environments. This paper presented experiments which allowed us to quantify this impact.

VI. Conclusion and Future works

With the increase in the number of Cloud computing users, Cloud providers have tackled challenges of how to attend growing demands in a scenario where the workloads within a virtual domain could not interfere the performance of other workloads running on the same physical hardware. This is a key motivation behind many works that explore performance isolation issues in many kinds of virtualization architectures. With the same motivation, we explored isolation issues of the LXC, which is the underlying container-based system where the Docker platform is settled. LXC is not the only container-based virtualization system that exists today. Google has launched the Lmctfy [45]—its system stack based on containers. OpenVZ [18] which is one of the oldest container-based system implemented for Linux and VServer [46] which introduces its capabilities into the Linux kernel in order to provide containerization features. Although there are a handful of possibilities for container-based Clouds. LXC is the one that become more popular in Cloud computing platforms due to its inclusion in source Kernel upstream.

The use of container-based virtualization systems under Cloud computing platforms has provided many benefits and a few drawbacks as we presented. However, an analysis of performance isolation intrinsic to these systems was an unexplored topic so far. Our results reveal that, unlike the KVM, the LXC does not provide complete isolation of resources for now. However, understanding the possible performance interferences, it is easy to conjecture several workload combinations to alleviate such interferences taking into account the impact they have among themselves.

The era of ”containerization” has begun in Cloud computing environments and the need for better understanding of the behavior of the workloads on these systems has become fairly desirable. Our work focused on evaluating performance interference suffered by disk-intensive workloads, but we plan to go further and study more in-depth these workload combinations on container-based Clouds. From what we know, best fitting them can contribute to greater user satisfaction and better resource usage that results in the end in a lower cost for maintaining infrastructure.

ACKNOWLEDGMENT

This work was supported by the Dell Inc.. All experiments presented in this paper were carried out using an infrastructure provided by Dell.

REFERENCES

[1] Amazon Web Services (AWS). [Online]. Available: http://aws.amazon.com

[2] Google Cloud. [Online]. Available: https://cloud.google.com/

[3] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[4] Linux containers (lxc). [Online]. Available: http://lxc.sourceforge.net

[5] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[6] R. Krebs, C. Momm, and S. Kounev, "Metrics and techniques for quantifying performance isolation in cloud environments," in *Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures*, ser. QoSA '12. New York, NY, USA: ACM, 2012, pp. 91–100.

[7] H. Fang, C. Li, Y. Wu, H. H. Huang, Z. Yang, and B. Zhao, "Understanding the effects of hypervisor i/o scheduling for virtual machine performance interference," in *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, ser. CLOUDCOM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 34–41.

[8] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 51–58.

[9] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 237–250.

[10] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments." in *ISPASS*, 2007, pp. 200–209.

[11] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao, "Who is your neighbor: Net i/o performance interference in virtualized clouds," *Services Computing, IEEE Transactions on*, vol. 6, no. 3, pp. 314–329, 2013.

[12] G. Casale, S. Kraft, and D. Krishnamurthy, "A model of storage i/o performance interference in virtualized systems," in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*. IEEE, 2011, pp. 34–39.

[13] 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 SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[14] KVM. [Online]. Available: http://www.linux-kvm.org

[15] 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), 2013 21st Euromicro International Conference on*. IEEE, 2013, pp. 233–240.

[16] M. G. Xavier, M. V. Neves, and C. A. F. D. Rose, "A performance comparison of container-based virtualization systems for mapreduce clusters," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. IEEE, 2014, pp. 299–306.

[17] VMware. [Online]. Available: http://www.vmware.com

[18] Openvz. [Online]. Available: http://www.openvz.org

[19] Control groups (cgroups). [Online]. Available: http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[20] D. R. Butenhof, *Programming with POSIX threads*. Addison-Wesley Professional, 1997.

[21] T. Krylosova, "Implementing container-based virtualization in a hybrid cloud," 2014, p. 37.

[22] S. Kim, H. Kim, J. Lee, and J. Jeong, "Group-based memory oversubscription for virtualized clouds," *Journal of Parallel and Distributed Computing*, vol. 74, no. 4, pp. 2241–2256, 2014.

[23] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: Challenges and approaches," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 55–60, Jan. 2010.

[24] 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*, ser. ExpCS '07. New York, NY, USA: ACM, 2007.

[25] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Proceedings of the 7th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 342–362.

[26] G. Kim, H. Park, J. Yu, and W. Lee, "Virtual machines placement for network isolation in clouds," in *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, ser. RACS '12. New York, NY, USA: ACM, 2012, pp. 243–248.

[27] A. Whitaker, M. Shaw, and S. D. Gribble, "Scale and performance in the denali isolation kernel," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 195–209, Dec. 2002.

[28] Y. Kanemasa, Q. Wang, J. Li, M. Matsubara, and C. Pu, "Revisiting performance interference among consolidated n-tier applications: Sharing is better than isolation," in *Proceedings of the 2013 IEEE International Conference on Services Computing*, ser. SCC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 136–143.

[29] Oracle. [Online]. Available: www.oracle.com

[30] MySQL. [Online]. Available: www.mysql.com

[31] Swingbench. [Online]. Available: http://dominicgiles.com

[32] Sysbench. [Online]. Available: http://sysbench.sourceforge.net

[33] S. T. Leutenegger and D. Dias, *A modeling study of the TPC-C benchmark*. ACM, 1993, vol. 22, no. 2.

[34] IOzone. [Online]. Available: http://www.iozone.org

[35] Ubuntu Linux Distribution. [Online]. Available: www.ubuntu.com

[36] K. Pepple, *Deploying OpenStack*. " O'Reilly Media, Inc.", 2011.

[37] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

[38] C. Waldspurger and M. Rosenblum, "I/o virtualization," *Commun. ACM*, vol. 55, no. 1, pp. 66–73, Jan. 2012.

[39] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "Eli: Bare-metal performance for i/o virtualization," *SIGPLAN Not.*, vol. 47, no. 4, pp. 411–422, Mar. 2012.

[40] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 275–287, Mar. 2007.

[41] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 51–58.

[42] A. Mandal, P. Ruth, I. Baldin, Y. Xin, C. Castillo, M. Rynge, and E. Deelman, "Evaluating i/o aware network management for scientific workflows on networked clouds," in *Proceedings of the Third International Workshop on Network-Aware Data Management*, ser. NDM '13. New York, NY, USA: ACM, 2013, pp. 2:1–2:10.

[43] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, "Performance analysis of network i/o workloads in virtualized data centers," *IEEE Trans. Serv. Comput.*, vol. 6, no. 1, pp. 48–63, Jan. 2013.

[44] R. Rosen, "Linux containers and the future cloud," *Linux J.*, vol. 2014, no. 240, Apr. 2014.

[45] Lmctfy. [Online]. Available: https://groups.google.com/forum/!forum/lmctfy

[46] B. des Ligneris, "Virtualization of linux based computers: the linux-vserver project," in *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*. IEEE, 2005, pp. 340–346.