Manuscript

Click here to download Manuscript cesga-paas.tex

Click here to view linked References

Journal of Grid Computing manuscript No.
(will be inserted by the editor)

Big Data-oriented PaaS architecture with disk-as-a-resource capability and container-based virtualization

Jonatan Enes · Javier López Cacheiro · Roberto R. Expósito · Juan Touriño

Received: date / Accepted: date

Abstract With the increasing adoption of Big Data technologies as basic tools for the ongoing Digital Transformation, there is a high demand for data-intensive applications. In order to efficiently execute such applications, it is vital that cloud providers change the way hardware infrastructure resources are managed to improve their performance. However, the increasing use of virtualization technologies to achieve an efficient usage of infrastructure resources continuously widens the gap between applications and the underlying hardware, thus decreasing resource efficiency for the end user. Moreover, this scenario is especially troublesome for Big Data applications, as storage resources are one of the most heavily virtualized, thus imposing a significant overhead for large-scale data processing. This paper proposes a novel PaaS architecture specifically oriented for Big Data where the scheduler offers disks as resources alongside the more common CPU and memory resources, looking forward to provide a better storage solution for the user. Furthermore, virtualization overheads are reduced to the bare minimum by replacing heavy hypervisor-based technologies with operating-system-level virtualization based on light software containers. This architecture has been deployed on a Big Data infrastructure at the CESGA supercomputing center, used as a testbed to compare its performance with OpenStack, a popular private cloud platform. Results have shown significant performance improvements, reducing the execution time of representative Big Data workloads by up to 4.5x.

Keywords Big Data · Platform as a Service (PaaS) · cloud computing · disk-as-a-resource scheduling · operating-system-level virtualization

Jonatan Enes · Roberto R. Expósito · Juan Touriño
Computer Architecture Group, Universidade da Coruña, Campus de A Coruña, 15701 A Coruña, Spain
Tel.: +34 881 011 212
Fax: +34 981 167 160
E-mail: {jonatan.enes,rreye,juan}@udc.es

Javier López Cacheiro
Fundación Centro de Supercomputación de Galicia (CESGA), Santiago de Compostela, Spain
E-mail: jlopez@cesga.es

1 Introduction

Big Data applications are being increasingly demanded by a wide range of users, from the scientific community to large corporations when the need for data analysis arises and other previous techniques like Data Mining or Data Warehousing are unfit for the task due to the huge size of the datasets. With the appearance of distributed processing frameworks like Apache Hadoop MapReduce [13] and Apache Spark [47], there has been a deluge of Big Data applications and associated technologies, which in turn have forced the existing infrastructure models to change in order to meet the new requirements like data volume, velocity and complexity [21].

Since their inception, Big Data clusters were built upon key concepts such as scalability and redundancy, in order to create systems that were able to directly grow as needed and to process the required volume of data. Because of this, these systems typically demand as underlying infrastructure a high number of interconnected nodes, very large and easily accessible storage space with high Input/Output (I/O) bandwidth and a network capable to handle the movement of very large datasets [3]. Unfortunately, current large-scale computing systems are mainly designed and oriented with either very specific goals in mind, like High Performance Computing (HPC) clusters, or with a broad set of users and requirements to cater to, like cloud computing infrastructures [8]. On the one hand, HPC systems focus on optimizing CPU and memory management for high demanding applications and thus they may lack the data storage requirements that Big Data needs. Furthermore, they usually expose the resources using batch-queueing job schedulers (e.g., SLURM [45]), an interface which has proved to be very difficult to bridge for Big Data applications [22]. On the other hand, cloud computing, or other similar service-oriented infrastructures, can provide the user with access to virtually infinite storage space and a high level of flexibility in regards to resource management or infrastructure size as a whole. However, to achieve this degree of flexibility, the very core of cloud computing is based on virtualization, which allows to separate a single physical machine into multiple virtual machines. Unfortunately, most current cloud services make heavy use of hypervisor-based virtualization, severely hindering the performance of network and disk I/O access [15, 20, 34].

This paper proposes a Platform as a Service (PaaS) architecture specifically oriented to Big Data that enhances I/O-intensive applications, where users can deploy clusters in the same easy way as a cloud platform but benefiting from better performance. With this proposed architecture as a testing ground we bring forward a novel approach for resource management and scheduling, adding customizable resources to be handled alongside traditional CPU and memory resources. In this work we will focus on providing the disk-as-a-resource capability. In addition, we also use the preferable lightweight container-based virtualization technologies. The combination of these two enhancements aims to bridge the gap between available platforms like HPC infrastructures or the cloud and the requirements that current and future Big Data applications may have as part of the Digital Transformation.

An implementation of this architecture has been successfully deployed on a real Big Data infrastructure installed at the Galicia Supercomputing Center (CESGA) [10]. Using this testbed, the performance of our PaaS architecture has been compared with a private cloud platform deployed using the popular OpenStack

framework [33], which does not offer disks as resources and by default makes use of hypervisor-based virtualization. This comparison shows a significant improvement of the execution times of real-world Big Data applications as the result of combining an efficient storage access and light virtualization.

The remainder of this paper is organized as follows: Section 2 describes the previous technologies, terms and concepts that are later used. The state of the art of the current platforms along with their limitations are described in Section 3. Section 4 presents the design of the proposed architecture and the main guidelines for its implementation, a high-level comparison between our solution and a private cloud platform, and a real case scenario deployed at CESGA. Following in Section 5, a more in-depth comparison is given with real measurements acting as the performance evaluation. Finally, Section 6 summarizes our concluding results.

2 Technical foundations and problem statement

To understand the key technical reasons that make interesting our approach, it is important to analyze the currently most common implementations of Big Data technologies and their architectures, as well as their limitations and the different ways of overcoming them. The focus will be on the high overhead they impose on storage access, which is considered critical for Big Data applications, and the overall high use they make of virtualization.

2.1 HPC architectures and virtualization technologies

From the HPC environment the management of two basic resources first appeared: CPU and memory. These two components are still today key resources and part of any scheduler or resource management system. Additionally, thanks to the evolution of hardware-assisted virtualization extensions from major CPU vendors (e.g., Intel VT-x, AMD VT-V), both resources can be easily abstracted, shared and used in an isolated way without incurring any significant performance penalty. Taking advantage of this, hypervisor-based virtualization technologies (e.g., Xen [4], KVM [23]) allow using and running many isolated instances commonly known as virtual machines. Virtual machines have been proven very useful throughout the years in many IT environments or even in HPC scenarios, where the performance penalty is affordable taking into account the flexibility provided [46]. Nevertheless, device virtualization overheads are still to be taken into account as they can degrade the performance of I/O-intensive applications significantly [15,16,34].

However, this traditional and efficient virtualization based on CPU, memory and virtual machines has been taken one step further with the adoption of operating-system-level virtualization, which basically eliminates a layer. With this technology, the role of the hypervisor disappears and the previous virtual machines are now translated into multiple isolated user-space instances that run together sharing the same underlying kernel. Such instances are often called software containers, virtual private servers, jails or zones. This kind of "light" virtualization makes the instances less demanding in CPU, memory and I/O requirements. Moreover, because the instances usually contain only the minimum processes required

for the service, resource requirements are even further reduced compared to virtual machines. Finally, thanks to the fact that these containers run next to the kernel, thus having direct access to the resources, they are able to achieve near bare-metal performance. Operating-system-level virtualization as a whole is still evolving but it is now an emerging alternative to be considered when performance is a priority [39].

One particular container-based virtualization technology that is currently being actively developed is Docker [26]. Docker allows users to launch lightweight containers by using the resource isolation features of the Linux kernel such as cgroups and kernel namespaces to isolate and provide resource limiting (i.e., CPU, memory, disk and network). This is important for security and service level agreement functionalities, so that even when sharing a single kernel and pool of resources, no container is able to overuse either the host's or other container's resources or have access to non-authorized resources. In [29], the authors provide an in-depth description of the operating-system-level virtualization and its requirements to be used in a real scenario. It is interesting to mention the special remark regarding the storage volumes used with container technologies. These technologies use layered file systems (e.g., OverlayFS) to provide more flexible management of containers, particularly for the root file system, and thus they are not intended to have high I/O performance [27]. So, the storage solution used to persist or handle data is left to the user's choice thanks to the possibility of binding volumes from the host to the container, the volume's storage back end being abstracted from the container. A good choice of back end is most important if write-intensive workloads are to be executed. Nevertheless, [27] does not consider host's disks as possible container volumes.

In our PaaS architecture, Docker has been chosen to provide the lighter and more resource efficient containers that can be later used to create instances that host an application or comprise a cluster. Moreover, Docker instances can be created with attached storage volumes as disks imitating traditional virtual machines. This feature will allow exposing dedicated underlying host disks to applications running inside containers.

2.2 Cloud-based architectures and services

Moving away from the low-level HPC model that seeks performance of the key resources above all (i.e., CPU and memory), there are the currently booming cloud-based architectures and services [25], where the focus is put on the service that is offered instead of its back end implementation or even its efficiency. Cloud services are very popular nowadays, being used from single users to large corporations, which now have the opportunity to outsource from software to even hardware in order to lower costs and consolidate services and infrastructure. There are many underlying technologies involved in cloud architectures that are beyond the scope of this paper, virtualization being one of them, but it is worth considering the level of affordability and flexibility that these platforms offer. Unfortunately, to achieve these features a high penalty is present, especially when taking into account the storage performance, which can become something not tolerable for Big Data applications that require to process huge amounts of data efficiently or within time limits. Several studies exist [7,36] that describe how the penalty imposed by

remote or virtualized storage access (typically the case when the cloud is used) severely affects data-intensive applications. In [36], the authors provide an in-depth study of the challenges of data-intensive computing in the cloud. Of particular interest for our scenario are the challenges that arise from resource sharing and the efficiency of storage systems, with proposed solutions such as Mesos [18] for resource management and alternative scheduling algorithms that take into account disks as resources to be also evenly shared between instances.

Nevertheless, cloud architectures are still relevant because they are flexible and easy to use. From this usability point of view, it is interesting to be able to easily manage instances (e.g., start, stop, connect to), and regarding flexibility, it is interesting the capability to define their individual resources (e.g., number of CPUs, amount of memory, number of disks). All of these features have been taken into consideration when implementing our solution and thus the proposed PaaS provides the user with a cloud-like interface where applications can be properly managed, while on the back end such applications avoid performance issues typically present on cloud architectures.

2.3 Big Data architectures

The low-level resource management of HPC environments and the flexibility of cloud architectures are combined in the Big Data model. The Apache Hadoop framework and its vast software ecosystem are the de-facto environment that most of current Big Data architectures rely upon. From its second release, Hadoop uses Yet Another Resource Negotiator (YARN) [42] as resource manager, which in turn uses software containers and aggregates the resources using nodes, virtual cores and memory, creating a cluster that exposes a pool of resources to the applications. The other core component of any Hadoop cluster is the Hadoop Distributed File System (HDFS) [38], which takes care of managing all the data by mainly distributing and replicating the files using its own file system. Together, YARN and HDFS create an architecture that provides the applications with an environment that offers scalability, flexibility and redundancy, while at the same time tries to achieve high performance. Nevertheless, not all Big Data applications need a framework like Hadoop and may be deployed in a standalone or semi-standalone way (e.g., an application that uses HDFS but runs independently). Some examples are easily found in applications such as NoSQL databases like Apache Cassandra or MongoDB, which run fully independently, or Apache HBase which uses HDFS. These applications can also benefit from the flexibility that the new platforms offer but may require a lower level approach with the resources, especially when it comes to storage efficiency [19].

However, when considering storage performance, it is especially interesting to mention a particular design pattern followed by all of the aforementioned applications, and by many other Big Data solutions. NoSQL databases or Hadoop HDFS do not directly use the local file system to store data, instead they rely on a custom or private file system that is created by joining a pool of directories and their underlying file systems in order to work. With this design feature, it is possible to scale the size and bandwidth of the underlying storage means by adding more volumes, ideally individually mapped to physical disks. This design guideline is based on the original view of a Big Data cluster as a commodity infrastructure

created from pooling together bare-metal nodes and hardware. Unfortunately, the benefits of this feature can not always be guaranteed when cloud environments are used, because of the storage and CPU virtualization penalties that may apply depending on the vendor's architecture design. Moreover, because the technical details of the architecture are usually hidden from the users as part of intellectual property, or merely because they may vary over time or across the infrastructure, it is not always possible to know beforehand if the penalties may or may not apply and to what extent. Nevertheless, our solution offers to I/O-intensive applications storage volumes that are directly mapped to dedicated disks, so that when a custom file system is created by joining the volumes, they can benefit from full disk bandwidth and thus improved performance over storage virtualization.

3 State of the art and related work

There are currently many companies that offer Big Data services and platforms in their catalog. However, their solutions usually share the infrastructure with other cloud-like services or products, which implies a heavy use of hypervisor-based virtualization in order to offer flexibility for the user and reduce costs in maintenance for the provider. This kind of virtualization certainly adds a significant overhead for storage resources and their underlying I/O performance, which are vital for Big Data frameworks and applications. This section analyzes some representative solutions, from infrastructure providers to Big Data frameworks, to identify the main problems they have. Finally, related work that tries to overcome such issues with different approaches is presented.

3.1 Platforms and architectures

When it comes to platforms and architectures, there are many available public cloud providers that offer the user the ability to deploy clusters or applications in just a few minutes, using a pay-per-use pricing model and providing a high degree of flexibility by allowing operations such as to create, destroy, resize or backup instances. Examples of such platforms are Amazon Web Services (AWS) [1], Google Compute Engine (GCE) [17] or Rackspace [30]. These services are commonly known as Infrastructure as a Service (IaaS) and have proven to be a flourishing model and business. While AWS and GCE use proprietary solutions for their back end infrastructure and the technologies they use are generally unknown, Rackspace on the other hand partially uses an open-source solution co-created with NASA, called OpenStack [33]. Nevertheless, for any major provider or platform it is certain that some degree of virtualization is used, which is particularly important for the management of storage volumes. A volume is an abstraction of a resource that represents a disk easily attached and detached from instances, backed up with snapshots and even moved around from one datacenter to another.

For specific and technical details on how the storage and the management of volumes are commonly implemented on these architectures, we can look at how OpenStack does it. OpenStack uses a component or software microservice, called Cinder, that handles volumes as block devices mapped on a pool of disks that are then hosted on the so-called block storage nodes, using underlying file systems such

as GlusterFS, Ceph or other proprietary solutions. The problem of this approach is that on top of the block storage access, which may be efficient, other software layers are placed to manage and achieve the previously described flexibility, which may add significant performance penalties [48]. Any additional layer means that the performance is reduced to some degree.

This limitation can be overcome, although at the cost of lower flexibility, if the disks are directly exposed to the instances running. The main goal of our platform is to offer the storage access as untreated as possible. So, the only additional treatment will be the one needed to assure that a disk offered as a resource is never shared between different running instances at the same time, both for security and performance issues. Finally, this idea of improving the efficiency of the instances is also further reinforced with the use of lightweight virtualization through software containers, as opposed to the use of hypervisor-based virtualization most likely used by providers like AWS and GCE.

3.2 Big Data frameworks

We can also look at the framework level with examples that host Big Data applications such as Apache Hadoop or Apache Mesos. In these two examples, schedulers and resource managers just expect the disks to be ready to use, with HDFS to create a custom, higher-level file system and with Mesos to create scratch or raw shared space. Both frameworks use disks to create a unified storage space that is then offered, while abstracting and never offering the disk itself as a resource. This forces giving away the control of the storage resources to the frameworks, which just use them to create and share the storage space with all the running applications, and thus the applications can only hope that the underlying disks provide efficient performance and that the disk access is evenly shared [35]. This scenario may not be important when such frameworks are deployed on a controlled and private cloud infrastructure, but when the framework (e.g., a Hadoop cluster) is deployed on an IaaS provider (e.g., AWS), the two limitations (i.e., virtualization of storage access and loss of disk control) may apply.

Fortunately, in order to improve flexibility in regards to resource management, especially in heterogeneous environments (i.e., where different kinds or versions of resources coexist), current schedulers like Mesos offer the feature to extend the type of resources and their amount exposed to applications. In our platform, to overcome the aforementioned loss of disk control, this extension would allow a scheduler to offer the disks as individual and accountable resources, implementing the disk-as-a-resource feature and making the scheduler "disk-aware". Moreover, this approach is suitable for other resources not commonly managed by schedulers such as Graphics Processing Units (GPUs) or to even differentiate resource types like faster hard drives (solid state, SSD) from slower ones (magnetic, HDD). In this paper, we present a Mesos extension to properly offer specific disks as resources that can be used only by one instance at the same time, thus adding them to the pool of resources, alongside CPU and memory, that can be customized for an instance.

3.3 Related work

Previous works on cloud storage systems, architectures and commercial platforms have proved that they are certainly attractive to end users and big companies alike with many benefits such as the cost effectiveness and its assured redundancy, with figures such as 99.99% availability [44]. However, there are even more studies that have also exposed the varying but significant performance penalty that any form of cloud-based storage imposes, the increasing differences between cloud providers or even between different services within the same provider [11,24].

The need to improve the performance of applications that run on cloud platforms has led to the appearance of some specific solutions and systems that target the very core of cloud architectures and their usually shared-hardware approach. Some works that aim to increase the efficiency of such architectures try to improve fairness when it comes to resource isolation [37], while others exploit this tenant approach with the ethically dubious so-called Resource-Freeing Attacks (RFAs) [41]. However, these solutions do not try to change the current model of cloud architectures or improve it, but rather propose a solution on top of it.

On the other hand, regarding lightweight virtualization with containers, there are studies and solutions that prove that the use of technologies such as Docker presents a significant improvement on resource usage, deployed by PaaS and Cloud vendors [5,14]. In [40], the authors have shown that a combination of Docker containers with a container management system like Amazon EC2 Container Service (ECS) can serve as the foundation to build a resource-efficient and at the same time flexible PaaS. However, disk management is delegated to volume services like Amazon Elastic Block Storage (EBS), which is a limitation for Big Data applications that need efficient storage. Moreover, even though Docker containers are used and managed using Amazon ECS, the underlying infrastructure still uses EC2, which is Xen-based and thus, in the end, container virtualization is placed on top of hypervisor-based virtualization. A recent related study has been conducted in [31], where parallel applications deployed on containers and using different managers including Mesos are compared to a cloud deployment with OpenStack and a bare-metal scenario. The results show that indeed Docker containers are closer to bare-metal when it comes to storage efficiency and have significantly lower overheads than the KVM hypervisor. Nevertheless, this work only focuses on computational overheads for deployment and execution of workloads, leaving aside the storage efficiency of the different scenarios, particularly if I/O-intensive applications are executed. Finally, in [9] the authors look forward to bring closer heterogeneous cloud environments by using an intermediate layer in the form of a PaaS. This layer is able to abstract the user from several underlying infrastructures and provide a more flexible user interface than an IaaS. Unfortunately, this work also leaves aside the disks as resources that can be managed and assigned to applications, probably considering that the orchestration APIs and back ends considered for the IaaS platform (e.g., OpenStack) do not directly allow for this feature.

4 Proposed PaaS architecture

In order to avoid the limitations previously exposed, mainly the overuse of virtualization and its special penalty on storage performance, this paper proposes a novel PaaS architecture with the disk-as-a-resource capability and the usage of lightweight software containers for the instances. Our solution is specifically designed for Big Data applications and aims to enhance the current cloud or PaaS models based on schedulers and resource managers by adding disks as another eligible resource. Its overall design and implementation follows a microservice-based architecture which is described in Section 4.1. The proposed solution is then compared with a private cloud architecture in Section 4.2 so as to highlight the main differences that stem from adding the disk-as-a-resource capability. Finally, a functional implementation of this architecture has been deployed on a real Big Data scenario at CESGA (Section 4.3).

4.1 Design and implementation

To deploy our platform, a basic underlying hardware infrastructure with some specific Big Data features and a software microservice-based architecture are needed. The main guidelines to create both are described next.

4.1.1 Underlying hardware infrastructure

The proposed architecture requires a commodity cluster of interconnected machines with no particular technologies or special requirements like low-latency networks generally used in HPC environments. In general, the infrastructure used for Big Data applications can be considered different to that used in HPC or cloud scenarios. Instead of focusing on a very high amount of CPU and large memory size, storage disks also become a key hardware resource.

If a general guideline has to be followed, it is preferable to use a high number of redundant nodes with a large amount of memory over the number of available CPUs, and more importantly, a high number of local disks. This is crucial taking into account that in our solution disks are offered as resources alongside CPUs and memory, instead of being shared. It is also worth noting that Big Data applications focus on data storage and speed (more disks) and data processing (more memory), which allows introducing two metrics when assessing a Big Data infrastructure: the disks-to-cores (i.e., disks/cores) and disks-to-memory (disks/memory) ratios. Examples of these metrics are presented later with a representative Big Data infrastructure.

4.1.2 Microservice-based architecture

Regarding the software, one of the main goals of our solution is to avoid any heavy virtualization or management overhead, especially with storage resources. Thus, it is key to look for resource managers and schedulers that are able to gather and understand the nodes as a pool of resources, are able to extend its functionality with usually unsupported resources such as disks, and are as well capable of deploying light software containers. For all of these purposes, Apache Mesos

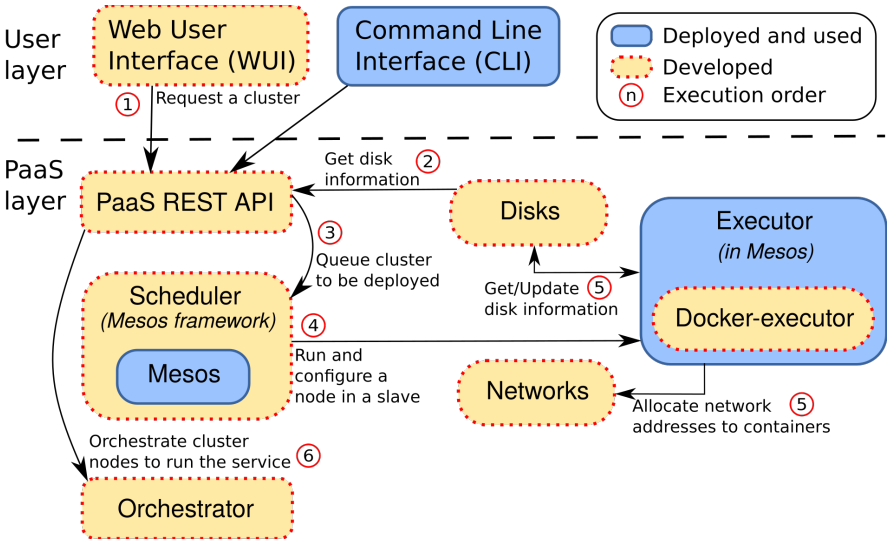


Fig. 1 High-level overview of the proposed PaaS architecture (each box in the PaaS layer represents a microservice)

has been chosen as the scheduler and resource manager, and Docker as the container manager. Additionally, other services were developed as part of our overall microservice-based architecture, as shown in Figure 1. Using this architecture for our PaaS, a functioning platform is created using components with defined functions, which in turn may be changed or further developed without affecting the whole system thanks to the use of interfaces. Complying with the microservice architecture model, smaller services are generally preferable. Therefore, technologies like Flask [32], a Python microframework with RESTful support, are used as the base for all the custom-made microservices that are described next.

Mesos is a key component of this architecture (see the *Scheduler* microservice in Figure 1). Using its extension features, which allow the administrator to define custom resources, the local disks on each node can be exposed as manageable resources that the frameworks and applications may ask for. This is the key feature that allows us to provide the disk-as-a-resource capability, although additional microservices had to be developed, as explained later. It is also worth mentioning that although Mesos supports the deployment of containers including Docker, this support is not enough to replicate the behaviour of other cloud computing services or platforms (e.g., attach or detach floating or host-independent IP addresses to the containers, or manage and mount the volumes onto host disks appropriately). This forces us to use a more flexible approach by using Mesos customizable executors, which are pluggable programs that handle job lifecycles. So, our own custom executor has been implemented (see *Docker-executor*), which in turn will handle Docker containers appropriately.

Other important microservices that have been implemented are: 1) *Networks*, which manages interfaces, networks and IP addresses later used in the instances to implement the host-independent networking that the containers use; 2) *Disks*, used to configure how the disk resources are exposed to the *Scheduler* and to account

Table 1 PaaS REST API product endpoints

PaaS Endpoints	
Endpoint	**Description**
GET '/products'	Get the catalog of applications
POST '/products/<product>/<version>'	Launch an application
GET '/clusters/<username>/<product>/ <version>/<id>'	Get information of an application
DELETE '/clusters/<username>/<product>/ <version>/<id>'	Stop and destroy a given application

which disks are used by what instances; and 3) *Orchestrator*, which handles the lifecycle of the individual cluster instances as well as of the final application that is later started on such cluster.

Finally, taking inspiration in many other cloud services, both a *REST API* and a graphical *Web User Interface* are added to make the platform usable for either end users or other services. Some of the endpoints exposed by the *REST API* are described in Table 1. These endpoints implement the main actions that the end user would need in order to use this PaaS, in a similar way to IaaS providers, such as: (1) visualize a catalog of possible Big Data applications; (2) launch one of the offered applications from the catalog; (3) get application information; and (4) destroy user applications. For the web user interface, AngularJS [12], a JavaScript framework for single-page applications, has been used.

4.2 Overall comparison with OpenStack

In order to highlight the novel approach of this architecture, this section compares our solution with OpenStack, a popular open-source platform for cloud computing, mostly deployed as IaaS. OpenStack is widely used to create private clouds and follows a similar approach to that of the big public cloud providers like AWS or GCE. Because of this, OpenStack also suffers from the performance penalties already discussed in Section 3.1.

To use OpenStack, the user mainly has to choose an OS image and a size of the instance or "flavor" (i.e., a combination of resources like virtual cores, memory and scratch storage as defined by the administrator). Once requested, this instance is scheduled and placed on an available compute node. In a similar way, all the disks used by the instance are created as volumes and likewise scheduled and placed on storage nodes (see Figure 2). It is important to note this split between compute and storage nodes. The differentiation of compute and storage nodes grants flexibility but it also means that volumes may not always be stored in the same node than the user instance, depending on the distribution of services and nodes chosen by the administrator.

This latter possible disassociation would be the first penalty imposed by OpenStack (see label 1 in Figure 2), as data have to be requested and retrieved from remote underlying hosts, which adds network overheads. Moreover, storage nodes

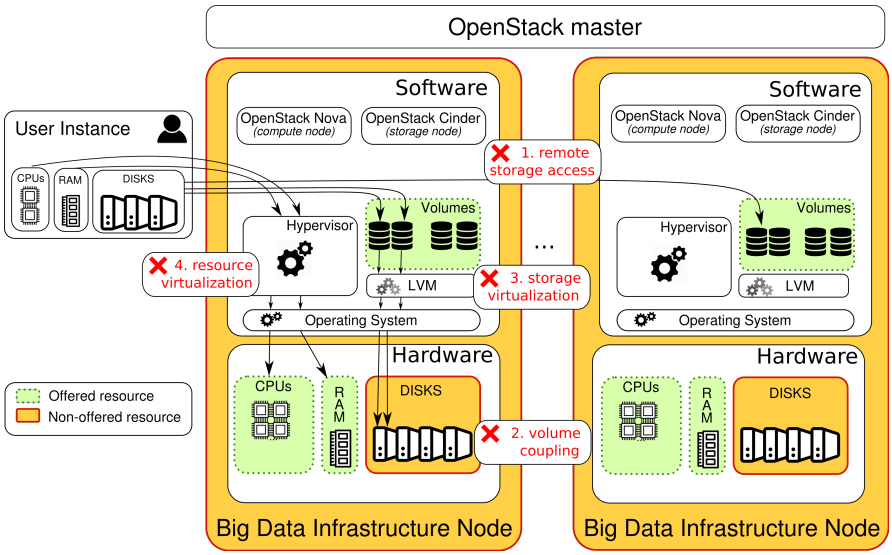


Fig. 2 Resources exposed and performance penalties with OpenStack

use technologies like Logical Volume Management (LVM) to host an unlimited number of volumes using a limited set of disks. Therefore, at some point two or more volumes will have to simultaneously use the same disk, sharing its I/O bandwidth, which is the second and most severe penalty when it comes to storage performance. The third penalty would be the use of intermediate layers like LVM for the storage virtualization, adding a constant overhead to any I/O operation. Finally, the fourth penalty would be the use of hypervisor-based virtualization, which imposes an overhead that although minor for CPU and memory, it is always present. As a whole, in the OpenStack architecture the user can only pick and configure the CPU and memory requirements for the deployed applications. Regarding storage volumes, only their size and number can be chosen, leaving their performance optimization and management to the administrator.

On the other hand, our PaaS architecture adds disks alongside CPU and memory to the pool of resources that the user can select from (see Figure 3). By using this disk-as-a-resource approach, the instances have dedicated access to the disks and can benefit from their full I/O performance. It has to be noted though that flexibility is reduced with our solution, as by design the data stored in the disks can only be accessed later by asking for the same node and disk. This loss of control over data placement is the most important trade off that the user has to make when using our platform. However, to mitigate this downside, it is encouraged to deploy applications when they are expected to be used for long periods of time before being destroyed. As a consequence, any data intended to be saved should be moved outside before the application is destroyed, in order to properly persist

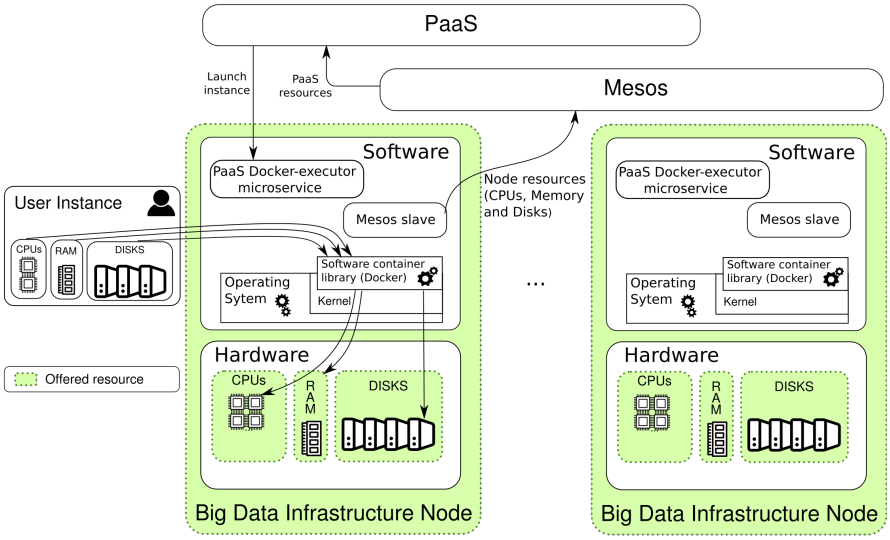


Fig. 3 Resources exposed with the proposed PaaS

Table 2 Main differences between our solution and OpenStack

	Big Data-oriented PaaS	**OpenStack**
Virtualization technology	container-based	hypervisor-based
Instance disks	directly mounted from local disks	virtual volume hosted locally or remotely
Instance configurable resources	CPU, memory and disks	CPU, memory and volumes

it in any on-premises storage solution close to the PaaS. To summarize, all the important differences discussed in this section are described in Table 2.

4.3 A real use case: CESGA's Big Data infrastructure

The deployment of our platform on a Big Data infrastructure is of great interest in order to assess and compare its performance with other cloud platforms. Recently, CESGA acquired a new hardware infrastructure specifically oriented to Big Data applications, being the ideal scenario to deploy and test our platform (as will be shown in Section 5).

This particular infrastructure is composed of two racks containing a total of 38 nodes, 4 of which have slightly different hardware (faster SSD disks) to play a role of master nodes if necessary. The main hardware and software specifications of both master and slave nodes are described in Table 3. In this real scenario, it is possible to measure the previously introduced ratio disks-to-cores (see Section 4.1.1), which would be 1:1 (12 disks:12 cores), and the ratio disks-to-memory which would roughly be 1:5400 MB (12 disks:64 GB).

Table 3 Big Data infrastructure at CESGA: hardware and software configuration

Hardware configuration	
#Nodes	38
CPU model	2x Intel Xeon E5-2620 v3 Haswell-EP
CPU speed	2.4 GHz (3.2 GHz in Turbo mode)
#Cores	12 (2x6)
Memory	64 GB DDR4
Network	1x10 Gbps + 2x1 Gbps
Disks	8x 480 GB SSD SATA 2.5" (master) 12x 2 TB NL SATA 3.5" (slave)
Software configuration	
Operating System	CentOS Linux release 7.2.1511
Java version	OpenJDK build 1.8.0_171-7.b10
Disk file system	XFS
Mount disks options	rw,noatime,nodiratimeattr2,inode64,noquota

5 Performance evaluation

One of the main goals of this paper is to prove that our container-based and disk-aware architecture may as a whole improve the overall performance of real-world Big Data applications compared to current private cloud-based platforms. This section presents the results of the performance evaluation of our architecture compared with OpenStack. First, Section 5.1 describes the experimental configuration and the benchmarks used in the evaluation. Next, performance results are analyzed in Sections 5.2 (low-level benchmarking) and 5.3 (application-level benchmarking).

5.1 Experimental configuration and methodology

For the performance evaluation, two different testbeds have been deployed on the CESGA Big Data infrastructure (see Table 3). The first one is our PaaS implementation and the second one is a private cloud deployed using OpenStack. The latest release of OpenStack (Queens) has been configured following the official guidelines of the OpenStack Foundation [28], representing the typical and default configurations for a standard deployment.

For each testbed, the experiments have been conducted at two different levels. On the one hand, the first series of experiments, carried out in Section 5.2, consist of assessing the raw I/O performance of a single instance using the Flexible I/O (FIO) tester tool [2]. These tests represent a low-level disk benchmarking that evaluates both sequential and random access using 8 independently mounted volumes working in parallel with representative configurations in terms of block size and Queue Depth (QD) (see Table 4). On the other hand, Section 5.3 analyzes the overall performance of Big Data applications on a 4-instance cluster with 8 volumes per instance (i.e., 32 volumes in total). It is worth noting that the cluster size of both testbeds was limited by the allowed use of the infrastructure and not by any scalability boundary. For the application-level experiments, two popular data processing frameworks, Hadoop (version 2.7.3) and Spark (version 1.6.3), were deployed and executed using the Big Data Evaluator tool (BDEv) [6, 43]. Two representative Big Data workloads (see Table 5) have been selected: (1)

Table 4 Configuration of FIO benchmarks

	Block size	**File size**	**Queue Depth (QD)**
Sequential Write/Read	4 KB, 1 MB	1 GB	1
Random Write/Read	4 KB	256 MB	1, 64

Table 5 Configuration of Hadoop/Spark benchmarks

Benchmark	**Description**	**Input size**
TeraSort	Sorting of an input dataset generated by TeraGen	120 GB
PageRank	Ranks websites by counting the number and quality of the links to each one	12 M of pages with 5 iterations

TeraSort, which is an I/O-bound benchmark which orders an input dataset; and (2) PageRank, an iterative CPU-intensive application which ranks websites. For both series of experiments the results provided take into account a minimum of 10 measurements, and variability is shown by including error bars in the graphs to indicate the minimum and maximum sample values.

It is very important when testing the performance of both a single instance or the 4-instance cluster, whether their volumes share the underlying disk access or not. In the case of our PaaS, instances are always given dedicated and local access to the disks and thus volumes directly map to disks. However, it is never sure in OpenStack if a volume will be placed in the same storage node where the instance is running (see label 1 in Figure 2). In all the scenarios presented for OpenStack, the remote volume access penalty has been manually removed by migrating and placing the volumes locally to the instance. Additionally, on a real scenario it can not be assured if the volume will be placed on the same storage node and disk as other instances' volumes, thus sharing the underlying disk access (see label 2). This latter penalty is of special interest and from now on it will be referred to as *volume coupling* or just *coupling*, defined by dividing the number of active instance volumes by the number of underlying disks used. Although there is support in OpenStack to balance the distribution of volumes across the storage nodes so that coupling is reduced using a combination of filters and weights to choose and order storage nodes and their disks, it is unavoidable that, in the end, multiple instance volumes share the same hardware resources.

Both the low-level and application-level benchmarks are evaluated using different levels of coupling. These different scenarios are compared against our PaaS, where neither coupling nor remote volume access penalties apply. Figure 4 shows the volume mapping layout for OpenStack. As can be seen, four different configurations have been evaluated, with varying coupling values from 1 (8 volumes share 8 disks) to 4 (8 volumes share 2 disks). Nevertheless, these OpenStack configurations have been artificially created to provide a comparison with our PaaS by using more favourable scenarios than in a real-world case, where the layout of the instance volumes can greatly vary over time as instances and volumes are created and destroyed, and the number of volumes backed by a disk can be greater than that of the worst case presented (see configuration 4 in Figure 4).

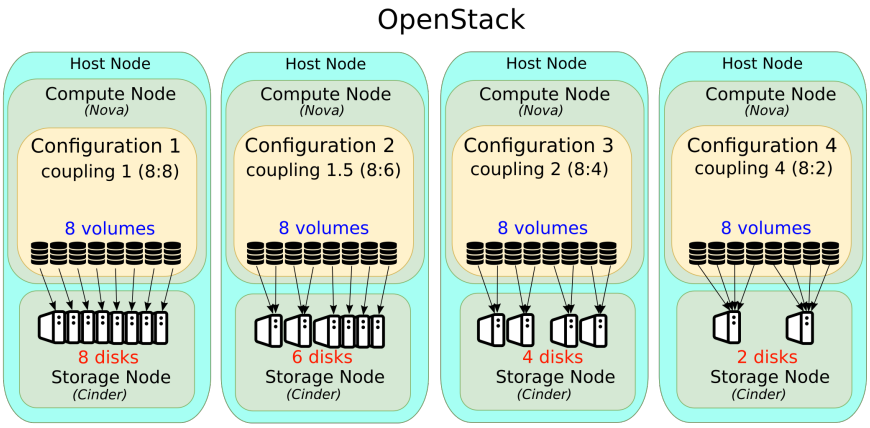


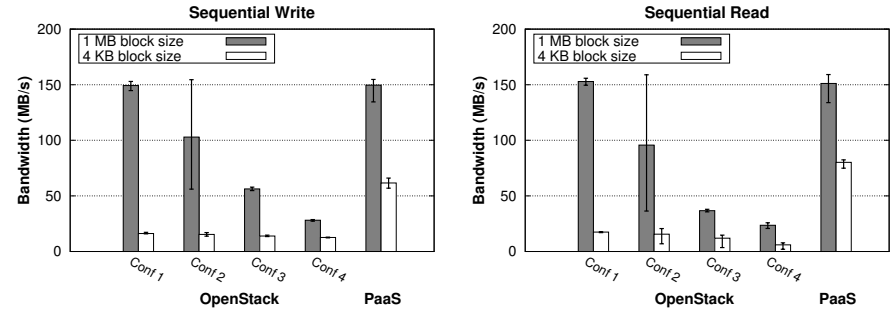
Fig. 4 Volume mapping configurations in OpenStack

5.2 Low-level disk benchmarking

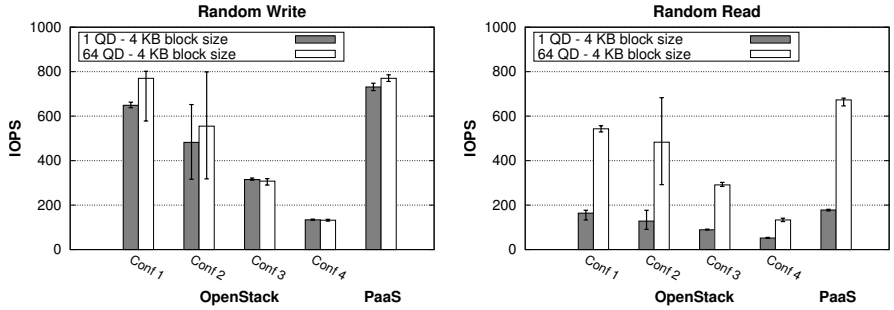
Figure 5(a) shows the sequential write and read bandwidths for our PaaS and for each OpenStack configuration detailed in Figure 4. The metric shown in the bars is the mean bandwidth of the 8 volumes, while error bars indicate the lowest and highest measures. These experiments basically write and read a single file per volume using a certain block size (i.e., the transfer size of each underlying I/O operation at the file system level).

In the sequential write scenario (left graph), the mean bandwidth for OpenStack is reduced as the coupling rises, especially when using a large block size (i.e., 1 MB). More specifically, it can be observed that the bandwidth is roughly halved from configuration 1 to 3 and 3 to 4 for 1 MB block size. In this case, PaaS performance is similar to OpenStack configuration 1, where coupling has been manually avoided. However, when considering a small block size (4 KB), the OpenStack performance is significantly lower than PaaS even in the best-case scenario (configuration 1). It is worth noting that, with this block size, coupling does not significantly affect the bandwidth, which can be explained by the fact that with a small block size more I/O transfers are needed and thus the virtualization overhead becomes the main bottleneck.

In the sequential read benchmark (right graph), the same bandwidth reduction trend is also observed according to the coupling. Our PaaS architecture again achieves a very similar bandwidth to OpenStack configuration 1 with a large block size, while it clearly outperforms OpenStack if a small block size is used. Note the high variability of OpenStack configuration 2 for both sequential tests when a 1 MB block size is used, the values being almost equal to configuration 1 and 3 for the highest and lowest measures, respectively. This is explained by the volume to disk mapping shown in Figure 4. In configuration 2, there are volumes that have dedicated disks, like configuration 1 has, while others have to share a disk with another volume, like configuration 3 does, thereby having simultaneously “fast” and “slow” volumes. This difference is not observed in any other cases as the volume mapping is balanced, and thus their variability is significantly lower.



(a) Sequential write and read results (in MB/s)



(b) Random write and read results (in IOPS)

Fig. 5 Performance results for the low-level disk benchmarking

Figure 5(b) presents the random write and read results in terms of I/O operations Per Second (IOPS) using a 4 KB block size, which is a representative testbed for replicating real-world scenarios (e.g., databases, Web servers). Both benchmarks show that a higher QD affects performance, this effect being more important in configurations with less contended volumes like 1 and 2, where coupling is low. This is explained by the fact that configurations with volumes mapped on dedicated disks can issue more I/O operations and expect them to be completed faster, thanks to the dedicated access, while the remaining configurations do not take advantage of the queue to offset the random operations, as the queue will fill soon. The high variability of configuration 2 is due to the same reason explained in the sequential scenario. Finally, our PaaS is able to improve OpenStack performance in all scenarios, showing higher bandwidths and IOPS than the most efficient OpenStack configuration.

5.3 Application-level benchmarking

Figure 6 presents the execution times of TeraSort and PageRank using Hadoop (left graph) and Spark (right graph) with the different OpenStack configurations and the PaaS, showing the median and the lowest and highest values. These experiments were conducted on a dedicated Hadoop cluster, without any interference

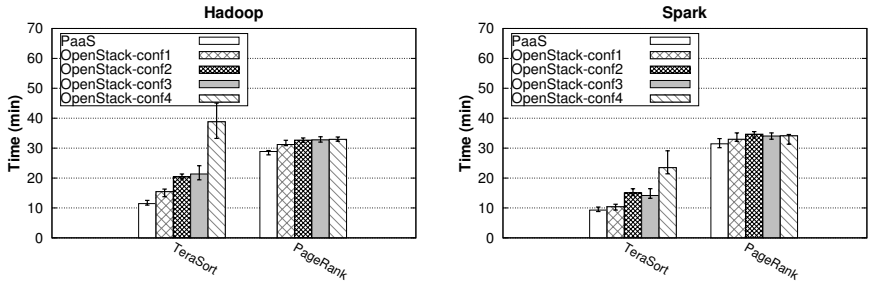


Fig. 6 Application-level performance results on an dedicated environment (in minutes)

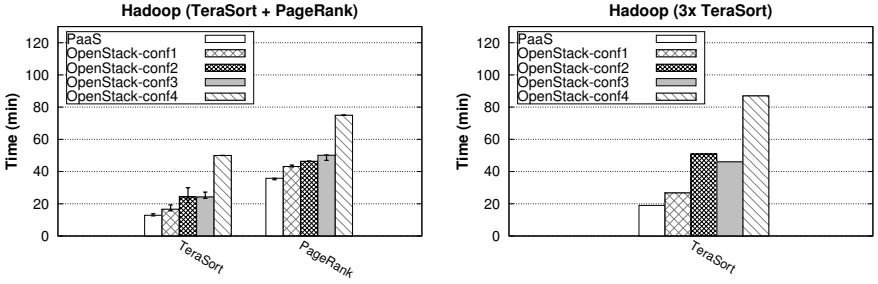
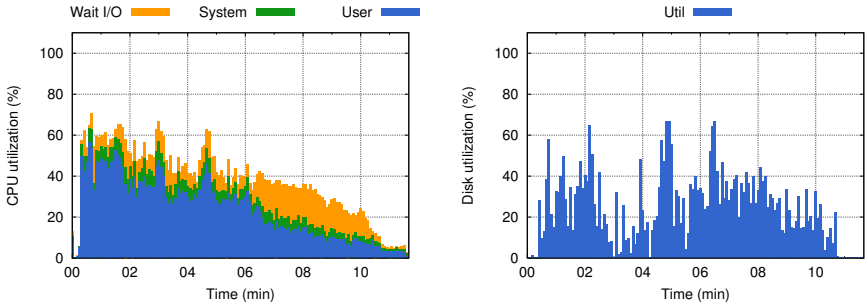


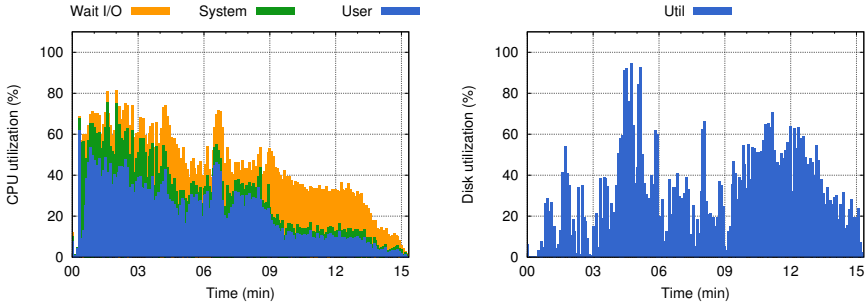
Fig. 7 Application-level performance results on a shared environment (in minutes)

between the execution of the workloads. Regarding TeraSort, it can be clearly observed that our PaaS deployment significantly outperforms OpenStack, specially with coupling, obtaining gains in Hadoop ranging from 32% (for the first configuration) to 239% (for the fourth configuration), and gains in Spark from 12% to 152%. The I/O-bound nature of this workload enables our PaaS to fully exploit the underlying disk bandwidth as shown in Figure 5(a), allowing to provide a speedup of up to 3.4x. This reinforces one of the main conclusions of this work, the fact that the storage efficiency of current platforms is vital for Big Data applications. Regarding PageRank, this iterative workload is mostly CPU-bound, with light disk I/O, and consequently presents moderate gain, exposing mainly the benefits of replacing the hypervisor-based virtualization with containers. The performance gain ranges from 8.1% (for the decoupled scenario) to 14.1% (for the fourth configuration) in Hadoop, and from 4.8% to 8.5% in Spark.

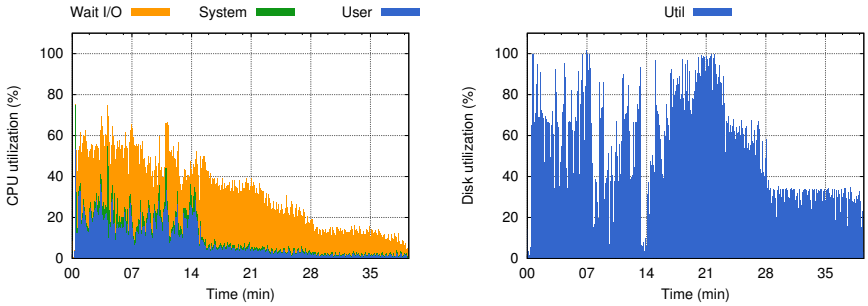
Figure 7 presents the results of running the TeraSort and PageRank workloads on a simulated shared environment, where different users launch their applications at different times but always forcing their execution to overlap at some moment. Only Hadoop results are presented, as the Spark ones are very similar. The left graph shows the overlapped execution of both TeraSort and PageRank by two different users. The right graph shows the results of executing three overlapped instances of TeraSort. For these experiments, the underlying available bandwidth is fully exploited, as with several different tasks submitting I/O requests the underlying disk accesses will be more randomly spread. In the first scenario, both benchmarks benefit from the PaaS with gains ranging from 29.5% (configuration 1) to 250% (configuration 4) for TeraSort, and from 20.5% to 85% for PageRank.



(a) PaaS



(b) OpenStack with configuration 1 (volumes decoupled)

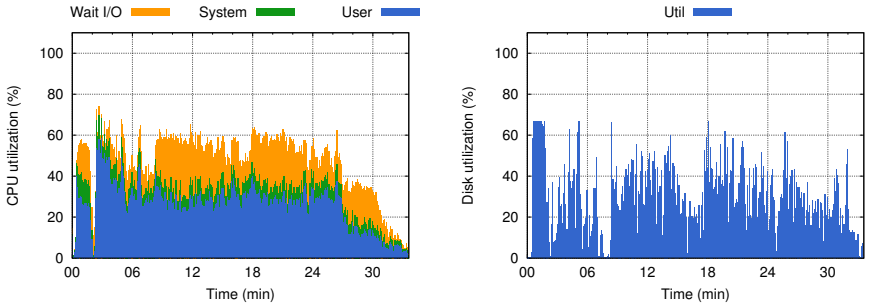


(c) OpenStack with configuration 4 (volumes coupled 8 to 2)

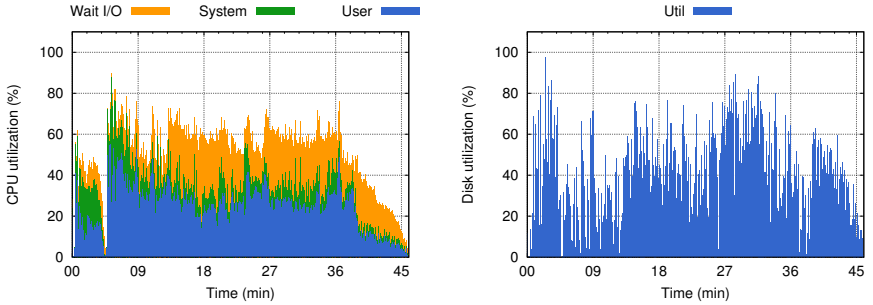
Fig. 8 CPU and disk utilization during the TeraSort execution on the dedicated environment

In the second scenario, the overlapped TeraSort executions further show the effects of volume coupling, as the gains go from 40.9% in configuration 1 up to 358% when configuration 4 is used. To summarize, in the shared environment the maximum gains achieved have a factor of about 3.5x and 4.5x for the first and second scenarios, respectively.

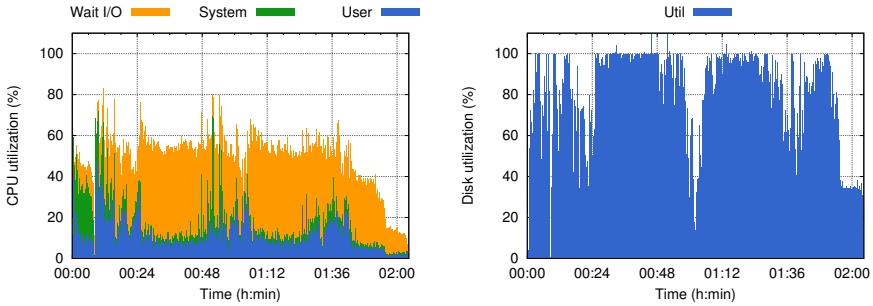
To provide more insights on the performance differences between the PaaS and OpenStack, Figure 8 shows the CPU (left plot) and disk utilization (right plot)



(a) PaaS



(b) OpenStack with configuration 1 (volumes decoupled)



(c) OpenStack with configuration 4 (volumes coupled 8 to 2)

Fig. 9 CPU and disk utilization during the TeraSort execution on the shared environment

during the execution of a single TeraSort using Hadoop on the dedicated environment. Figure 9 presents the same graphs when simultaneously executing three TeraSort workloads using Hadoop on the shared environment. For the disk plots, a single disk is represented from the pool of all disks available to the instances, after having checked nonetheless that the usage is spread more or less evenly across all disks. From top to bottom, the first row shows the PaaS results and the second and third rows the OpenStack results with configurations 1 (volumes decoupled)

and 4, respectively. It can be appreciated in the CPU utilization that a significant percentage of execution is spent waiting for I/O operations, as expected by the heavy I/O nature of TeraSort. However, this percentage is clearly higher for OpenStack, further increasing with the volume coupling. In OpenStack's configuration 4 particularly, the amount of time spent in I/O wait is even larger than the amount of time spent in user execution, a situation that shows the heavy penalty that coupling imposes. Regarding disk utilization, it can be seen how TeraSort was not actually affected by any disk bottleneck on the dedicated environment, unlike the several TeraSort instances simultaneously executed on the shared environment. In this latter scenario, disk utilization percentages increase and while the PaaS does not cause any bottleneck, OpenStack begins to show high utilizations with clear bottlenecks in configuration 4.

6 Conclusions

Current Big Data applications still lack proper support in the most commonly used platforms, architectures and cloud providers. Aside from HPC systems, most of the available solutions were not designed with performance in mind but with flexibility and good usability. However, this search for a high level of flexibility has led to an increasing use of virtualization technologies, which allow providers to abstract the resources and the underlying infrastructure from the applications that use them. In particular, storage is a resource that is currently heavily abstracted, generally considering its efficiency of lesser importance than computing power provided by CPU and memory. Typically, the most common penalties that affect storage resources are the underlying resource sharing without proper policies to avoid it, the use of remotely accessible storage, and the virtualization overheads imposed by volume managers or intermediate layers. Unfortunately, most Big Data frameworks and applications rely on storage and its underlying I/O performance to execute efficiently. Thus, it is critical that newer platforms, especially those with Big Data uses in mind, pay special attention to storage resources and the way they are handled and scheduled.

In this paper, we have proposed a novel PaaS solution specifically targeted for Big Data applications, which uses a microservice-based architecture that allows the scheduling of disks as resources so that applications can use them in a fully dedicated manner and exploit their underlying I/O bandwidth. Moreover, Docker containers were used, considering the current movement that aims to replace traditional hypervisor-based virtualization with a lighter approach based on operating-system-level virtualization in the form of containers. This combination of improvements over traditional architectures is interesting to be considered when storage performance is critical, as proven by the comparison with OpenStack, using representative Big Data workloads. The results obtained on a real Big Data infrastructure deployed at CESGA have shown that when virtualization overheads are removed or mitigated and, most importantly, applications and frameworks are given bare-metal disk bandwidth, performance can be significantly improved in real-world scenarios with multiple workloads running simultaneously. This PaaS architecture is available for CESGA users at `https://bigdata.cesga.es`.

The architecture proposed in this work could be improved in terms of flexibility and usability. Future work includes reusing the data stored on the disks, as data is

neither moved nor deleted, by allowing the user to do an exact instance mapping in the same nodes and disks if the cluster or application has been stopped or destroyed. Fortunately, the scheduler and the underlying layers (e.g., Mesos) allow this remapping in a straightforward way by looking and asking for specific disks. Additional planned features could also leverage Mesos to allow dynamic resizing of clusters both vertically (i.e., resources of the instances) and horizontally (i.e., number of instances), and to give the user the option to choose from different types of resources (e.g., HDD or SSD disks) that are available throughout the PaaS.

Acknowledgements This work was supported by the Ministry of Economy, Industry and Competitiveness of Spain (Project TIN2016-75845-P, AEI/FEDER, EU), and by the FPU Program of the Ministry of Education (grant FPU15/03381).

References

1. Amazon Web Services (AWS): `https://aws.amazon.com/`. Last visited: June 2018
2. Axboe, J.: FIO tool github site. `https://github.com/axboe/fio`. Last visited: June 2018
3. Bakshi, K.: Considerations for Big Data: architecture and approach. In: IEEE Aerospace Conference, AeroConf'12, pp. 1–7. Big Sky, MT, USA (2012)
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: 19th ACM Symposium on Operating Systems Principles, SOSP'03, pp. 164–177. Bolton Landing, NY, USA (2003)
5. Bernstein, D.: Containers and cloud: from LXC to Docker to Kubernetes. IEEE Cloud Computing **1**(3), 81–84 (2014)
6. Big Data Evaluator (BDEv): `http://bdev.des.udc.es/`. Last visited: June 2018
7. Bryk, P., Malawski, M., Juve, G., Deelman, E.: Storage-aware algorithms for scheduling of workflow ensembles in clouds. Journal of Grid Computing **14**(2), 359–378 (2016)
8. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6), 599–616 (2009)
9. Caballer, M., Zala, S., García, Á.L., Moltó, G., Fernández, P.O., Velten, M.: Orchestrating complex application architectures in heterogeneous clouds. Journal of Grid Computing **16**(1), 3–18 (2018)
10. CESGA Supercomputing Center website: `http://www.cesga.es/`. Last visited: June 2018
11. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: 1st ACM Symposium on Cloud Computing, SoCC'10, pp. 143–154. Indianapolis, IN, USA (2010)
12. Darwin, P.B., Kozlowski, P.: AngularJS web application development. Packt Publishing (2013)
13. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008)
14. Dua, R., Raja, A.R., Kakadia, D.: Virtualization vs containerization to support PaaS. In: IEEE International Conference on Cloud Engineering, IC2E'14, pp. 610–614. Boston, MA, USA (2014)
15. Expósito, R.R., Taboada, G.L., Ramos, S., González-Domínguez, J., Touriño, J., Doallo, R.: Analysis of I/O performance on an Amazon EC2 cluster compute and high I/O platform. Journal of Grid Computing **11**(4), 613–631 (2013)
16. Ghoshal, D., Canon, R.S., Ramakrishnan, L.: I/O performance of virtualized cloud environments. In: 2nd International Workshop on Data Intensive Computing in the Clouds, DataCloud-SC'11, pp. 71–80. Seattle, WA, USA (2011)
17. Google Compute Engine (GCE): `https://cloud.google.com/compute/`. Last visited: June 2018
18. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R.H., Shenker, S., Stoica, I.: Mesos: a platform for fine-grained resource sharing in the data center. In: 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI'11, pp. 295–308. Boston, MA, USA (2011)

19. Jacobs, A.: The pathologies of Big Data. Communications of the ACM **52**(8), 36–44 (2009)
20. Ji, C., Li, Y., Qiu, W., Awada, U., Li, K.: Big Data processing in cloud computing environments. In: 12th International Symposium on Pervasive Systems, Algorithms and Networks, I-SPAN'12, pp. 17–23. San Marcos, TX, USA (2012)
21. Kaisler, S., Armour, F., Espinosa, J.A., Money, W.: Big Data: issues and challenges moving forward. In: 46th Hawaii International Conference on System Sciences, HICSS'13, pp. 995–1004. Wailea, HI, USA (2013)
22. Katal, A., Wazid, M., Goudar, R.H.: Big Data: issues, challenges, tools and good practices. In: 6th International Conference on Contemporary Computing, IC3'13, pp. 404–409. Noida, India (2013)
23. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: KVM: the Linux virtual machine monitor. In: Ottawa Linux Symposium, OLS'07, pp. 225–230. Ottawa, Canada (2007)
24. Li, A., Yang, X., Kandula, S., Zhang, M.: CloudCmp: comparing public cloud providers. In: 10th ACM Internet Measurement Conference, IMC'10, pp. 1–14. Melbourne, Australia (2010)
25. Mell, P., Grance, T.: The NIST definition of cloud computing. Communications of the ACM **53**(6), 46–51 (2010)
26. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux Journal (239), 76–91 (2014)
27. Mizusawa, N., Nakazima, K., Yamaguchi, S.: Performance evaluation of file operations on OverlayFS. In: 5th International Symposium on Computing and Networking, CANDAR'17, pp. 597–599. Aomori, Japan (2017)
28. OpenStack Installation Tutorial for Red Hat Enterprise Linux and CentOS: `http://docs.openstack.org/newton/install-guide-rdo/`. Last visited: June 2018
29. Peinl, R., Holzschuher, F., Pfitzer, F.: Docker cluster management for the cloud - survey results and own solution. Journal of Grid Computing **14**(2), 265–282 (2016)
30. Rackspace website: `https://www.rackspace.com`. Last visited: June 2018
31. Ramon-Cortes, C., Serven, A., Ejarque, J., Lezzi, D., Badia, R.M.: Transparent orchestration of task-based parallel applications in containers platforms. Journal of Grid Computing **16**(1), 137–160 (2018)
32. Ronacher, A.: Flask, a Python microframework. `http://flask.pocoo.org/`. Last visited: June 2018
33. Sefraoui, O., Aissaoui, M., Eleuldj, M.: OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications **55**(3), 38–42 (2012)
34. Shafer, J.: I/O virtualization bottlenecks in cloud computing today. In: 2nd Workshop on I/O Virtualization, WIOV'10, pp. 5:1–5:7. Pittsburgh, PA, USA (2010)
35. Shafer, J., Rixner, S., Cox, A.L.: The Hadoop distributed filesystem: balancing portability and performance. In: IEEE International Symposium on Performance Analysis of Systems & Software, ISPASS'10, pp. 122–133. White Plains, NY, USA (2010)
36. Shamsi, J., Khojaye, M.A., Qasmi, M.A.: Data-intensive cloud computing: requirements, expectations, challenges, and solutions. Journal of Grid Computing **11**(2), 281–310 (2013)
37. Shue, D., Freedman, M.J., Shaikh, A.: Performance isolation and fairness for multi-tenant cloud storage. In: 10th USENIX Symposium on Operating Systems Design and Implementation, OSDI'12, pp. 349–362. Hollywood, CA, USA (2012)
38. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop Distributed File System. In: IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST'10, pp. 1–10. Incline Village, NV, USA (2010)
39. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In: 2nd ACM European Conference on Computer Systems, EuroSys'07, pp. 275–287. Lisbon, Portugal (2007)
40. Tihfon, G.M., Park, S., Kim, J., Kim, Y.M.: An efficient multi-task PaaS cloud infrastructure based on Docker and AWS ECS for application deployment. Cluster Computing **19**(3), 1585–1597 (2016)
41. Varadarajan, V., Kooburat, T., Farley, B., Ristenpart, T., Swift, M.M.: Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In: 19th ACM Conference on Computer and Communications Security, CCS'12, pp. 281–292. Raleigh, NC, USA (2012)
42. Vavilapalli, V. K. et al.: Apache Hadoop YARN: Yet Another Resource Negotiator. In: 4th Annual Symposium on Cloud Computing, SOCC'13, pp. 5:1–5:16. Santa Clara, CA, USA (2013)

43. Veiga, J., Enes, J., Expsito, R.R., Tourio, J.: BDEv 3.0: Energy efficiency and microarchitectural characterization of big data processing frameworks. Future Generation Computer Systems **86**, 565–581 (2018)
44. Wu, J., Ping, L., Ge, X., Wang, Y., Fu, J.: Cloud storage as the infrastructure of cloud computing. In: International Conference on Intelligent Computing and Cognitive Informatics, ICICCI'10, pp. 380–383. Kuala Lumpur, Malaysia (2010)
45. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: Simple Linux Utility for Resource Management. In: 9th Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP'03, pp. 44–60. Seattle, WA, USA (2003)
46. Younge, A.J., Henschel, R., Brown, J.T., Von Laszewski, G., Qiu, J., Fox, G.C.: Analysis of virtualization technologies for high performance computing environments. In: 4th IEEE International Conference on Cloud Computing, CLOUD'11, pp. 9–16. Washington DC, USA (2011)
47. Zaharia, M. et al.: Apache Spark: a unified engine for Big Data processing. Communications of the ACM **59**(11), 56–65 (2016)
48. Zeng, W., Zhao, Y., Ou, K., Song, W.: Research on cloud storage architecture and key technologies. In: 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ICIS'09, pp. 1044–1048. Seoul, South Korea (2009)