The Integration of CloudStack and OCCI/OpenNebula with DIRAC

Víctor Méndez Muñoz[1,5], Víctor Fernández Albor[2,6], Ricardo Graciani Diaz[3,7], Adriàn Casajús Ramo[3], Tomás Fernández Pena[4], Gonzalo Merino Arévalo[1] and Juan José Saborido Silva[2]

[1] LHC Tier-1 Computing Production. Port d'Informació Científica (PIC). Campus Universitat Autònoma de Barcelona (UAB). Bellaterra, Spain.
[2] Particle Physics Department. Universidade de Santiago de Compostela (USC). Santiago de Compostela, Spain.
[3] Department of Structure and Constituents of Matter. Universitat de Barcelona (UB). Barcelona, Spain.
[4] Centro de Investigación en Tecnoloxías da Información (CITIUS). Universidade de Santiago de Compostela (USC). Santiago de Compostela, Spain.

Abstract. The increasing availability of Cloud resources is arising as a realistic alternative to the Grid as a paradigm for enabling scientific communities to access large distributed computing resources. The DIRAC framework for distributed computing is an easy way to efficiently access to resources from both systems. This paper explains the integration of DIRAC with two open-source Cloud Managers: OpenNebula (taking advantage of the OCCI standard) and CloudStack. These are computing tools to manage the complexity and heterogeneity of distributed data center infrastructures, allowing to create virtual clusters on demand, including public, private and hybrid clouds. This approach has required to develop an extension to the previous DIRAC Virtual Machine engine, which was developed for Amazon EC2, allowing the connection with these new cloud managers. In the OpenNebula case, the development has been based on the CernVM Virtual Software Appliance with appropriate contextualization, while in the case of CloudStack, the infrastructure has been kept more general, which permits other Virtual Machine sources and operating systems being used. In both cases, CernVM File System has been used to facilitate software distribution to the computing nodes. With the resulting infrastructure, the cloud resources are transparent to the users through a friendly interface, like the DIRAC Web Portal.

The main purpose of this integration is to get a system that can manage cloud and grid resources at the same time. This particular feature pushes DIRAC to a new conceptual denomination as *interware*, integrating different middleware. Users from different communities do not need to care about the installation of the standard software that is available at the nodes, nor the operating system of the host machine which is transparent to the user. This paper presents an analysis of the overhead of the virtual layer, doing some tests to compare the proposed approach with the existing Grid solution.

License Notice: Published under licence in Journal of Physics: Conference Series by IOP Publishing Ltd.

[5] Corresponding author: vmendez@pic.es
[6] Corresponding author: victormanuel.fernandez@usc.es
[7] Corresponding author: graciani@ecm.ub.es

1. Introduction
The DIRAC project was initially designed, in 2003, as a community management software to support scientific computing in the LHC context [1]. At that time, the specific use case was the Monte Carlo (MC) simulation of the LHCb experiment. Nowadays, DIRAC is used by LHCb as the management middleware of the community, including not only MC, but all the other computing activities such as the reconstruction of the experiment data collection, the selection of events by different filters, the reprocessing of that data, the user analysis or the data transfers. Moreover, other scientific communities besides LHCb are using DIRAC such as ILC, Belle II and others. Some of them have extended the DIRAC core functionality to fulfill their requirements [2]. These experiences have been useful to validate two important characteristics of the software: the robustness of the framework, and an extensible design, resulting in a solid software stack able to deal with different use cases in a flexible manner.

Additionally, DIRAC offers different tools to provide an integral solution for workload and data management on distributed computing infrastructures. From the beginning of the project, the need to provide a user friendly interface in the context of Virtual Organizations (VOs) was clearly identified. To fulfill this need, DIRAC provides a Web portal offering a friendly multi-user and multi-VO GUI, matching the different users operations, rights and permissions, including the computing administration in different layers, the final user analysis, or the anonymous view of the public domain. The DIRAC portal also offers the possibility to integrate other web portals. Using the DIRAC Web portal one can deal with the overlaying middleware in a transparent way. DIRAC was designed to make use of Grid and non-Grid technologies. For example, at the beginning of the project in 2003, the distributed resources were only available to LHCb via direct submissions to the Local Resource Manager Systems (LRMS). Later, the Grid technologies evolved towards offering Grid interfaces for remote access to batch queues.

For these reasons DIRAC can be defined as an *interware* that allows interoperability among different computing resources via different drivers to each of the middleware solutions. In this paper, a work aiming to extend the DIRAC interoperability between virtual computing infrastructures is presented. Starting from the previous work with the Belle experiment using Amazon Elastic Compute Cloud (EC2) [2], it is shown a description of the design, implementation and testing to integrate DIRAC with two well known Cloud management solutions: the specific CloudStack [3] and the Open Cloud Computing Interface (OCCI) [4], which is an open standard adopted by many Cloud manager providers like OpenNebula [5] or OpenStack [6].

This paper is organized as follows: Section 2 describes the previous DIRAC Virtual Engine to support the Amazon EC2 Cloud, Section 3 presents the additional CloudStack integration design. Section 4 is about the design of the OCCI integration with DIRAC. Section 5 includes the Virtual Image Engine of the CloudStack and the OCCI integrations, with detailed description of the particular aim and design of the two approaches. The Virtual Image Engine refers to the Virtual Machine (VM) platforms, Operating Systems, Virtual Instance contextualization, and software distribution taking advantage of the CernVM File System (CVMFS) [7] with different deployments depending on the particular purposes. Section 6 shows the tests of the CloudStack and OCCI/OpenNebula integration. The CloudStack Network of the Galicia Supercomputing Center (CESGA) has been used as the CloudStack testbed. The OCCI/OpenNebula testbed deployed at Port d'Informació Científica (PIC), aims to demonstrate the feasibility of such types of cloud resources to be used for the LHCb MC production. An analysis of the different overheads of the virtual layers, both CloudStack and OpenNebula, is shown in this section. Finally, a brief summary is presented in Section 7 together with an outlook of the future work in the Cloud interoperability and production deployment.

2. DIRAC Virtual Engine

The DIRAC integration in CloudStack and OCCI/OpenNebula is based in the previous extension of DIRAC to use Amazon EC2 [2], which has been proven as a robust and extensible software to provide computing resources for the Belle scientific community on the Amazon EC2. In this section, the main components of the design and implementation are described, providing the basic Virtual Machine functionalities to design and integrate DIRAC with CloudStack and OCCI/OpenNebula.

There are two server and two VM components, their mutual relations and their interactions with other DIRAC components are shown in Figs. 1 and 2. In these figures, the services/servers items have been colored in red, the client and interfaces in yellow, the DIRAC agents in green, and the databases in purple.

Virtual Machine Scheduler.
This component gets the list of tasks to be executed from the central Task Queues, by matching the pending tasks to the capabilities of the defined images. If not enough VMs are available and the maximum VMs threshold is not reached, then it submits a new VM using the specific VM Director.

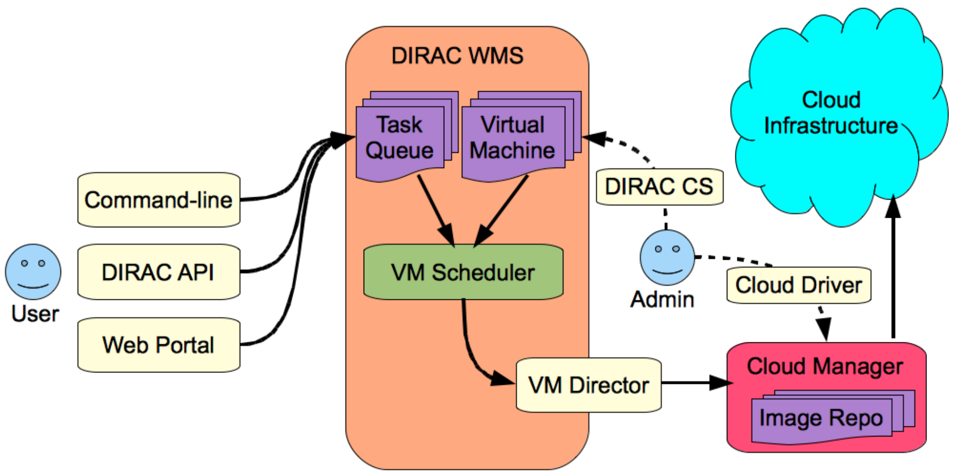

Figure 1. A generic Virtual Machine Submission in DIRAC

Fig. 1 shows the general interaction of the VM Scheduler with other components to submit a VM to a generic Cloud Manager, which can be Amazon EC2 or an OCCI compliant, i.e. OpenNebula. The DIRAC Admin has previouslly uploaded the image to the Cloud Manager using the corresponding Cloud Driver, and set these values on the DIRAC Configuration Server (CS).

The Virtual Machine Scheduler starts a full VM with DIRAC pre-installed and configured to execute the Job Agent, together with a VM Monitor Agent.

Virtual Machine Manager Service.
The VM Manager is part of the DIRAC Workload Management System (WMS). It acts as a middleware integration layer between different virtualization solutions. On one hand, it allows the interaction with other components. Furthermore, it contains the logic flow of the VM scheduling engine. On the other hand, this component is responsible for the persistence of the VMs information, through an associated database. Images are described in the DIRAC Configuration Service (CS) [8]. For these Images, the following parameters must be assigned:

1. *MinCPU*, is the minimum amount of CPU time required that will trigger the instantiation of a new VM; 2. *MinLoad*, is the minimum load required on the VM. This parameter is sent to the VM Monitor process running inside the instantiated VM and, when the load drops below this threshold for a certain time period, the VM is halted; 3. *Flavor*, is the resource provider for which the Image is valid, i.e., Amazon. The same image can be reused under multiple flavors and/or cloud providers; 4. *MaxVMs*, is the maximum number of VMs allowed to run simultaneously in a given resource provider, and 5. *Requirements* that is the dictionary defining the computing capabilities of the Job, used to match tasks.

Virtual Machine Monitor Agent.
This component executes on the VM, starting immediately after the start up of the operating system. Its first action is to declare the VM running state in the VM Manager Service component.

If there is an error in the connection or if any abnormal condition is reported back by the Manager, the VM is halted. The VM Monitor periodically monitors the CPU load of the VM, the number of executed tasks and the amount of output data of the VM. This information is reported to the VM Manager in heartbeats.

Finally, if the VM Monitor detects that the load of the VM has dropped below a certain configurable threshold, the VM is halted by the VM Manager Service if there are no pending transfers ongoing.

Virtual Machine Job Agent.
The DIRAC Job Agent does the matching of the tasks from the central TaskQueue and supervises their correct execution on the VM resource. It reports periodically the activity and task actions in case of abnormal behavior conditions.

This component connects to the DIRAC WMS matcher to request a new task pending for execution. All the connections are secured using the DISET mechanism [8] and server certificate-key pair available in the VM, previously declared in the DIRAC configuration.

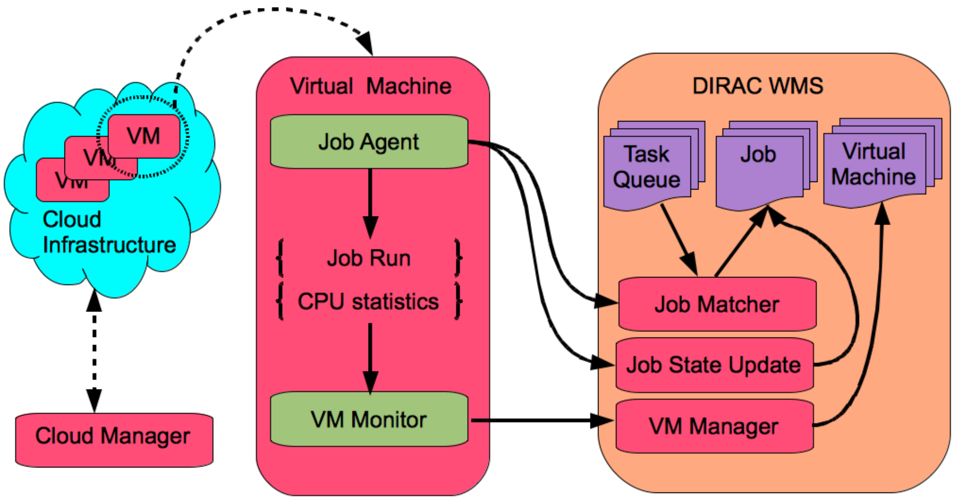

Figure 2. A generic Virtual Machine Running Job in DIRAC

In Fig. 2 the Job Agent and the VM Monitor Agent are interacting with the DIRAC WMS. The communication between the two agents of the VM is indirect. The VM Monitor gets system statistics to learn about different states of the Job such as stalled or finished and it informs the VM Manager. On the other hand, the Job Agent is interacting with the WMS for the Job matching, and managing different states such as *downloading input sandbox*, *really running* or *job completed*. This functionality of the DIRAC Job Agent is the same, independently of whether it is running in a VM or in a standard job in a real Worker Node.

3. Specific CloudStack Integration Design

This section describes the integration between DIRAC and CloudStack [9]. Both, the CloudStack internals and necessary additions to DIRAC are described in the next subsections.

3.1. CloudStack Architecture Overview

CloudStack [3] is an open-source cloud platform, written in Java, that allows building any type of Cloud including public, private, and hybrid. CloudStack was originally developed by Cloud.com, but now it belongs to the Citrix Systems company. Any deployment of CloudStack has five types of components, as shown in Fig. 3: Computer Nodes, Clusters, Pods, Availability Zones, and the Management Service.

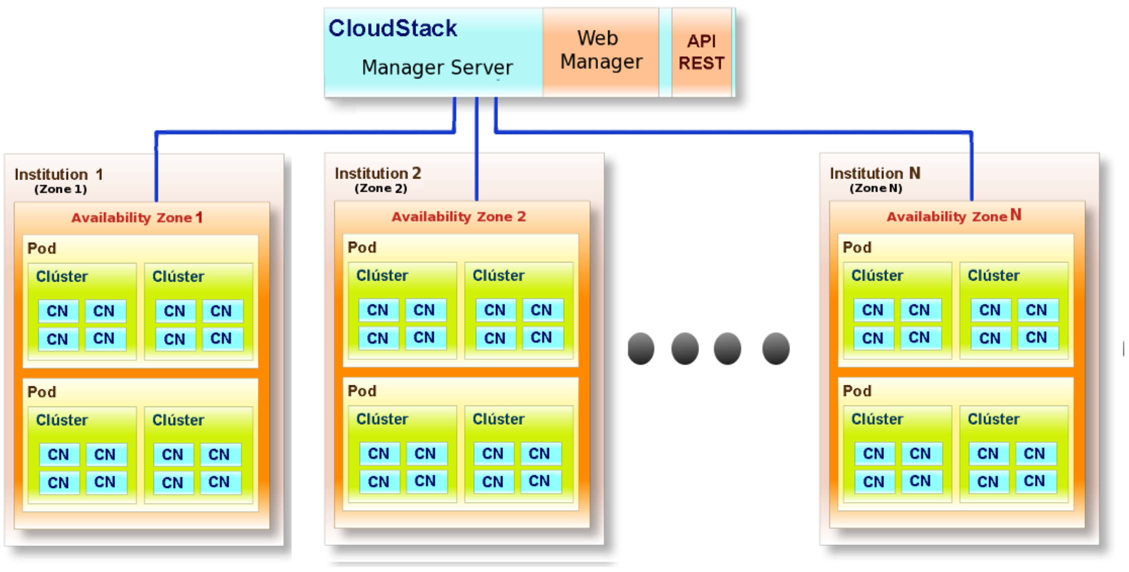

Figure 3. CloudStack Architecture

The Computer Nodes (CNs) are the resources that have the CloudStack agent installed, plus one of the hypervisors supported by the platform, like KVM, XenServer or VMware vSphere. These nodes allow the execution of VMs, being transparent for a third party. A group of CNs is a Cluster. All the CNs of a Cluster have the same hypervisor installed and share the same primary storage. A collections of Clusters is named a Pod. An Availability Zone is composed of a collection of Pods offered as IaaS. Finally, the Management Service allows to manage the entire Cloud.

The CloudStack platform can manage three roles: the Root Administrator, which has the highest access privileges, so it can manage the entire cloud, including the hosts, clusters, CloudStack user accounts, domains (an institution infrastruture), service offerings and templates (image templates); the Domain Administrator, which is able to perform administrative operations regarding one domain, but cannot access physical servers or other domains, and, the CloudStack user, which is an unprivileged role, so it can only manage its own virtual resources such as its VMs. DIRAC is accessing the infrastructure with the role of a CloudStack user.

CloudStack provides two interfaces: a Web interface and a RESTful API. Both components are used in the interaction with DIRAC. The RESTful API allows the communication with the DIRAC Server, and through the Web interface it is possible to create the credentials needed to access to the API.

3.2. DIRAC parameters for CloudStack
The description of the new templates is included in the DIRAC configuration with some parameters which define the interaction and the behavior of the system. The *API key* parameter is the key to add to the signature in order to verify the SHA1-mac, and the *Secret key* parameter to sign the previous message of the API before verifying the SHA1-mac. The *Name* parameter is used to identify the name of the template. In order to identify the OS Type that best represents the OS of this template, the parameter *Ostypeid* is used. The *Volumeid* parameter allows to identify the ID of the disk volume of the template. The format for the template (QCOW2, RAW, VHD) is identified by the *Format* parameter. The *Hypervisor* parameter allows to identify the target hypervisor for the template. With the *URL* parameter, the DIRAC Server knows where the template is hosted. And, finally, the *Zoneid* parameter allows to identify the ID of the zone of the template.

3.3. CloudStack DIRAC driver
This component provides all necessary methods to make calls to the CloudStack API as it is shown in Fig. 4 (VM Node Tasks). It is an important part in this implementation of DIRAC, because it is the interface between DIRAC and CloudStack. Together with this component, some generic RESTful of interest beyond this particular case.

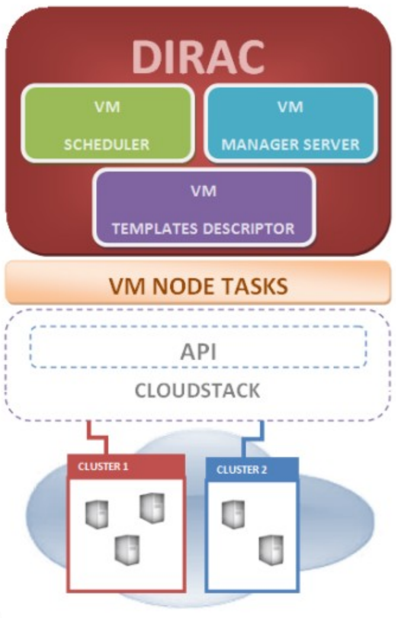

Figure 4. CloudStack Integration with DIRAC

The CloudStack DIRAC driver is useful to other components like the Scheduler component.

3.4. CloudStack Templates Descriptor
This component provides the application logic of the template creation which get the necessary parameters from the Dirac Configuration in order to create a new specific template to launch new

VMs .The VM Manager server uses the VM Templates Descriptor to take care of the persistence of the information of the newly created templates.

4. Specific OCCI/OpenNebula Integration Design

Initial DIRAC Virtual Engine design [2] was using the Amazon EC2 driver to manage the Cloud. The integration with OpenNebula could take advantage of such interface using the OpenNebula EC2. The aim is to provide as much connectivity as possible to different Cloud managers. Thus, this work is focussed in the OCCI interface provided by OpenNebula, which is the Cloud manager interface proposed as a standard by the Open Grid Forum (OGF). In this way, the DIRAC connectivity is enhanced from the *de facto* Amazon standard, with the OCCI Open-source community standard, as well as the CloudStack web interface described in Section 3, covering a broad spectrum of the available Cloud managers.

This section first explains some extensions to enable OCCI management in DIRAC and, second, it describes the DIRAC and OCCI/OpenNebula interaction and communication between the different components.

4.1. OCCI extensions to DIRAC

The extensions have been designed and implemented in the following classification

OCCI Client Interface.
The OCCI standard is a RESTful API. A Python interface has been developed which uses the OpenNebula OCCI client in order to encapsulate the OCCI request and the XML parsing of the OCCI replies.

Client Extensions.
Some of the components are subclasses of previous ones. Therefore, the OCCI Image and the OCCI Instance are subclasses of the VM Image and VM Instance, adapted to the use of the OCCI client. In the same way, the OCCI Director is a subclass of the Virtual Machine Director. The OCCI Director is part of the VM Scheduler agent used to submit new Virtual Machines to the OCCI Cloud manager. For this purpose, the OCCI Image client it is used, which can match the Configuration Server options for different Images and multiple flavors of each Image. In this way, a single VM Image can be named with different configurations per Cloud provider of the Image. The OCCI Instance gives functionality to the bulk management of the Instances.

Server Side Extensions.
The VM Scheduler Agent has been extended to include the OCCI Director Submit Pool. The VM Manager Service includes the corresponding RPC to the OCCI VM side requests. For this purpose, the DIRAC secure RPC framework is used, allowing certificates or proxies authorization, depending on the service configuration [8].

Virtual Machine Side Extensions.
Two Agents are running in the VM side. The Job Agent is configured to match the corresponding requirements of the OCCI VM Image. The VM Monitor has been extended to deal with the OCCI instance initialization and halting.

4.2. DIRAC and OCCI/OpenNebula Interaction and Communication

There are two basic stages of the specific design to integrate OCCI/OpenNebula with DIRAC: the VM submission and the Job running.

The OCCI VM submission follows the generic scheme shown in Fig. 1. The VM Scheduler Agent has a general simple algorithm as it was explained. The DIRAC WMS uses the

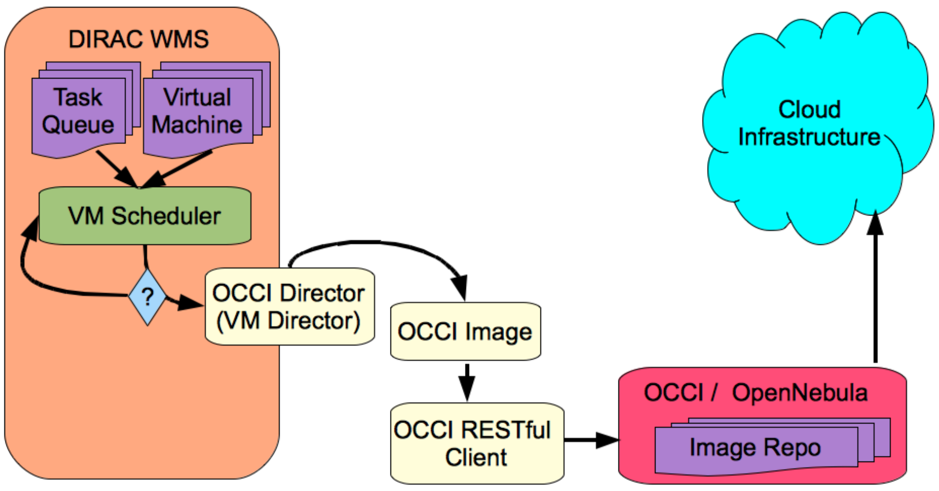

Figure 5. OCCI integration details of the VM submission

corresponding director to execute an OCCI compliant request to the OCCI server. Fig. 5 shows the details of the components interaction for the OCCI submission.

The second stage is the execution of the task in the VM, whose generic scheme was shown in Fig. 2. The Job Agent mechanism has already been described in Section 2. It is worth to mention that the VM Monitor takes care of the OpenNebula specific way of dealing with the VM ID and halting the Instance, as opposed to Amazon EC2 or CloudStack. The VM ID is got from the contextualization process as it is explained in Section 5.1. In OpenNebula, halting a running instance can not be fully achieved from inside the VM. Therefore, the VM Monitor has to trigger an OCCI standard halt request from the VM Manager

5. Image Engine

This section describes the image engine for the two approaches. The CloudStack approach aims to potential use cases which may need different OSs and thus different Virtual Images, while the OCCI approach is focused in CernVM images with an HEPIX contextualization [10]. Ideally one would like to have one golden image per supported OS and a powerful contextualization.

5.1. Image Engine in CloudStack

The instance creation of CloudStack is similar to the Amazon EC2 service. The CloudStack platform allows the administrator to define Service Offerings and Disk Offerings. These allow the administrator to define the virtual hardware (CPU speed and count, RAM size, and disk size) that the user can select when creating a new instance. The administrator can also provide templates, which are the base OS images that the user can select when creating a new instance. In the first version of the platform the user will only be able to call (from the DIRAC server) instances which were previously created. There is work in progress on a second version which will make possible the dynamic creation of instances from the DIRAC Server.

5.2. CernVM Image Engine in OCCI/OpenNebula

CernVM is a Virtual Software Appliance designed to provide a complete and portable environment for developing and running LHC data analysis on any end user computer and batch node, independently of the host operating systems. It contains only a minimal operating system required to bootstrap and run the experiment software. The experiment software is delivered from CERN repository using the CernVM File System (CVMFS) that decouples the operating system and the software area. CernVM allows contextualization using the HEPIX standard [10]. The CernVM instances are automatically managed by the DIRAC OCCI development as already explained, including creation, status report, and halting.

5.2.1. HEPIX Context Images. The DIRAC integration with OCCI/OpenNebula is dealing with the following images:

- A CernVM batch node. The CernVM image specifically designed to Grid and Cloud computing.
- On the fly configuration image. On the submission of a new VM, an HEPIX Context section is specified to provide the environment variables. VMID (the current VM ID) and necessary variables for the CVMFS client and the network configuration. The OCCI cloud manager creates a temporary ISO image with this environmental variables, and HEPIX booting system includes them in the system.
- An ISO context image. It is used by the CernVM HEPIX compliant booting system to configure the contextualization. First, it obtains the DIRAC VM service certificate and key for the DIRAC VM agents. Second it triggers the CVMFS client repository configuration and the network configuration. This configuration is done before the network services are started (prolog.sh script in the ISO). The last booting init process of the CernVM HEPIX (epilog.sh script in the ISO) is in charge of configuring and starting the DIRAC Virtual Machine Agents.

5.3. CVMFS

CVMFS is a file system for distributed environments which has the advantage that can be mounted as a normal file system through File in the User Space (FUSE) [7]. It was the software distribution mechanism used on both CloudStack and OCCI/OpenNebula integrations. In the first case, a complete server and client CVMFS has been deployed, including the software repository. The tests for the OCCI/OpenNebula integration are making use of the production LHCb CVMFS repository, deploying the client on the VMs.

5.3.1. The native CVMFS CERN repository. As it has been already mentioned, the DIRAC integration with OCCI/OpenNebula is targeting the LHCb use case. The configuration of the CVMFS cache hierarchy in these tests has been the standard one deployed in the LHCb production environment:

- local batch node memory
- local batch node disk cache
- cloud site squid proxy servers
- CERN distributed stratum CVMFS mirror servers
- CERN central repository

CVMFS offers several repositories for the different experiment software. For the tests described in this work, the lhcb.cern.ch repository has been used in order to run Monte Carlo LHCb applications.

5.3.2. CVMFS repository deployment and performance bechmarking. In order to have a test infrastructure as realistic as possible based on the CVMFS repository, an heterogeneous test environment has been deployed, consisting of virtualized nodes through CloudStack in the Galician Supercomputing Center (CESGA), and normal nodes in the LHCb TIER-2 cluster of the University of Santiago de Compostela (USC). The main components of the test infrastructure are, at the USC TIER-2, a CVMFS repository served by Tomcat on a machine of the LHCb cluster and a Squid HTTP proxy, which is important to check the scalability of the solution. An additional machine with a proxy server is also installed at CESGA to improve scaling.

To test the possible overhead caused by CVMFS when executing typical High Energy Physics (HEP) analysis jobs, a testbed environment was deployed based on the ROOT system (http://root.cern.ch).

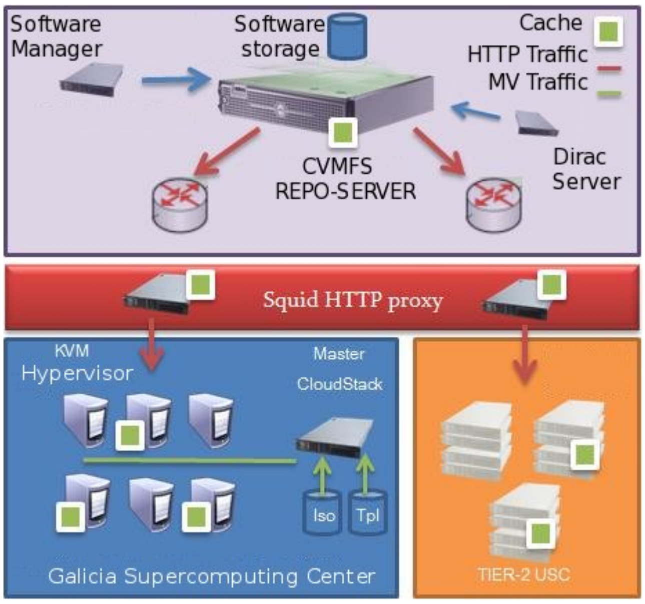

Figure 6. CVMFS Repository and Architecture Deployment

Fig. 6 shows the repository deployment and the rest of the CVMFS architecture for testing the loss of performances in a worst case scenario for the CVMFS, which can then be compared with the direct download, as detailed below. Table 1 shows some of the properties of CVMFS and the ROOT system. In particular, the number of files and the cache size.

Table 1. File Properties for ROOT and CVMFS

Filesystem	Size (MB)	#Files	Avg. Size (KB)
`ROOT`	218 (56 .tgz)	4630	2.5
`CVMFS Cache`	69	628	4.0

The tests show a comparison of CVMFS versus Direct Download (Tar File) from the same HTTP server. The VMs used are based on the KVM hypervisor, which is known to have about a 5% of performance reduction due to the virtualization overhead. To make a reliable comparison from a known and reproducible state, all caches are cleared up in every test iteration (the so-called cold-cache method). After a specified number of iterations are performed, the one which provided the worst results is chosen. The aim is to measure the loss of performance in a realistic case where a typical user has never launched any job in the nodes beforehand. This seems the most realistic simulation because it can be compared as well with the direct downloads tests, where the user is not going to own the cached software when she downloads the files.

The first algorithm in the Table 2 describes the application logic used to tests the CVMFS performance and scalability in terms of execution time. And the second algorithm shows the application logic used to test the Direct Download execution times in some scaling setups.

Table 2. Testing algorithms of the software area

Algorithm 1: CVMFS Performances

```
Input: Node list
Output: Time results

if { multicore execution } then;
    run iterate in each node/core
    time A ? mount CVMFS
    set env
    time B ? run rf202 extendedmkfit
if { node execution finish } then
    get results()
umount ? CVMFS
    delete ? Squid cache in each subnet
    delete ? CVMFS cache in each node
return [ worst result ]
```

Algorithm 2: Direct Download Performances

```
Input: Node list
Output: Time results

if { multicore execution } then;
    run iterate in each node/core
    time A ? download ROOT
    set env
    time B ? run rf202 extendedmkfit
if { node execution finish } then
    get results()
    delete ? Squid cache in each subnet
return [ worst result ]
```

Each of the tests was performed several times to compare all the results and to verify the errors of network saturation or peak network, which could have affected them. It was taken into account the impact of the bandwidth in wide area networks, where there is a higher latency than in local area networks.

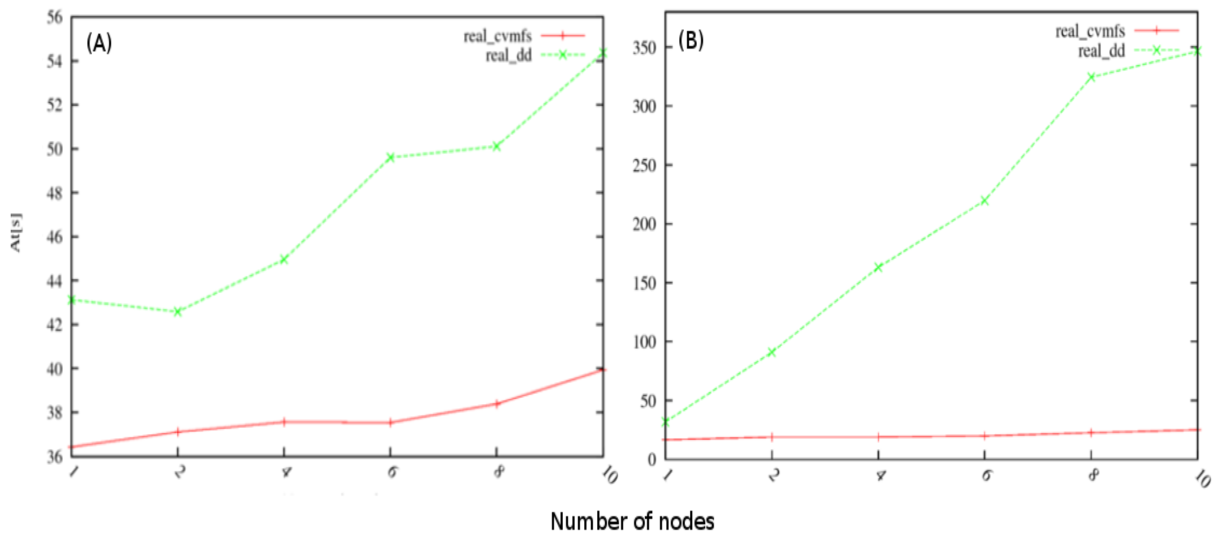

Figure 7. CVMFS comparison with Direct Download: (A) execution time of core/job in non virtualized nodes, (B) execution time of core/job in virtualized nodes.

The plots of Fig. 7 show on the left a test of a non virtualized nodes and on the right a test on virtualized nodes environment. The comparison between virtualized and non virtualized nodes has no sense, since they are heterogenous environments, with different computing power, different network bandwidth, and additionally, the non virtualized results are perform in a non dedicated production environment, while the virtualized is a dedicated environment. These plots shows two tests in order to compare the CVMFS and the direct download in two different environments.

The left plot of Fig. 7 shows the real execution time of tests in non virtualized nodes as a function of the number of nodes, for two different software repository infrastructures: CVMFS (red line) and direct HTTP download (green line). The execution time when using the CVMFS repository is substantially smaller than the one using direct HTTP download. The latter also increases strongly with the number of software requests, showing a clear advantage in favour of the CVMFS repository for a big number of simultaneous jobs running in the system. The low overhead of CVMFS caused by the software downloading needed to run the user's job could still be further reduced in a realistic situation, because usually that software is already cached locally by previously run jobs. This test (left plot) was performed in the same local area network, therefore there is not latency effect.

The right plot of Fig. 7 shows the execution time of tests in virtualized nodes. When using the HTTP download method (green curve) the time increase is proportional to the number of nodes, reaching about 6 minutes in the worst case (10 nodes). On the other hand, the curve corresponding to the CMVFS solution shows no significant execution time increase.

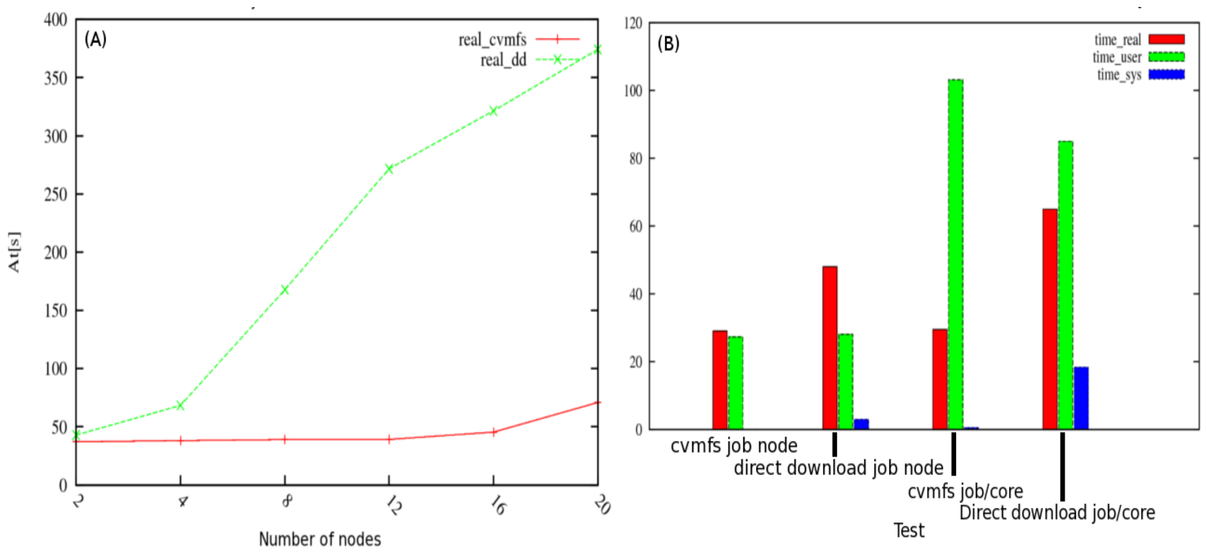

Figure 8. CVMFS Scalability on the Node Increasing and on the Remote Access: (A) Execution time of job/core in normal and virtualized nodes (B) Several types of software execution from USC CVMFS Repository to Barcelona University Cluster.

The left plot of Fig. 8 shows the tests execution time in environments composed of a mixture of normal and virtual nodes. Again, for the the HTTP download method the time increase is proportional to the number of nodes involved in the test, while the CVMFS shows only a very moderate increase when going from 16 to 20 nodes. This time increase is due to network congestion. The right plot of the same figure shows the execution time (per node and per core) for four different tests. In this case, the time increase is due to the latency of wide area networks, and it roughly doubles the execution time when going from CVMFS to HTTP download.

6. Test Results

This section includes different test for the CloudStack and OCCI/OpenNebula integration with DIRAC, in order to evaluate the two approaches design targets.

6.1. The CloudStack Testbed

In this first integration the testbed was the virtualized CloudStack Network of Galicia Supercomputing Center. The designed tests were running successfully in the infrastructure.

In the development environment, the first tests of this integration were the submission of simple jobs via the DIRAC User Interface (UI) for several VO users. The results were successful. A small number of failed jobs were sometimes seen. The origin of these failures was traced to come from the shutdown of some VMs by CloudStack, and as result, the jobs running in these VMs failed. In the next release of this prototype, those problems should be solved by identifying the cause of the failures, and resubmitting the failed jobs, or migrating the instances in the case of the infrastructure were needed for other purposes.

6.2. The OCCI/OpenNebula Testbed

The DIRAC integration with OCCI/OpenNebula has been tested using an LHCb application workflow. The main steps of the testing jobs are:

- Environment login configuration (LbLogin)
- Environment Gaudi and Gauss configuration (SetupProject)
- Monte Carlo event generator for LHCb (Gauss event generator application based on the Gaudi framework)

The test job is therefore a scientific application with high CPU consumption and low I/O.

The test infrastructure is the PIC cloud testbed infrastructure, which is using OpenNebula cloud manager and is deployed in 16 physical nodes. In this test, one node is used as OpenNebula cloud manager server, implementing the OCCI server. The other 15 nodes are cloud nodes with KVM hypervisor. Each physical node has 2 quad-core processors IntelXeon X5355 @ 2.66GHz each and total of 16GB memory.

A DIRAC service including the VMDIRAC extension integrating the OCCI/OpenNebula has been deployed. The virtual bootstrap image used for this tests is a cernvm-batch-node 2.4.0.

OpenNebula instantiate VMs of many different sizes. For this test, small instances with 1 core and 2GB of memory have been chosen. This is considered a *worst possible case scenario* aimed to define a baseline of performances for comparison purposes.

The test design is composed of 500 jobs of 100 events submitted to the DIRAC server. The DIRAC server is configured to run a maximum of 80 VMs on the OpenNebula cloud infrastructure, following the scheduling schema described in section 4.2. Every 60 seconds, the VM Scheduler agent will check the TaskQueue, and if a matching job is found, it will submit a new VM. Thus, having all the workload on the TaskQueue at the beginning of the test, the ramping-up of 80 VMs and continuous execution of the DIRAC Jobs can be tested. When the 80 VMs have been started, a total of 10 cloud nodes will be used, each running 8 VMs

One of the goals of the test is to obtain an overhead analysis of the OCCI/OpenNebula submission system and the HEPIX contextualization, and also, an overhead of the DIRAC machinery. To analyze the KVM hypervisor overhead, the metric of event/core/hour and the running job timestamps (walltime) were recorded and compared with the same metrics on jobs running in a non-virtualized environment. These are 50 jobs of 100 events directly submitted to CREAM CE, and running in a single Worker Node with exactly the same hardware that the cloud nodes in the OpenNebula setup and with a standard WLCG production configuration (Scientific Linux 5 OS with a CVMFS client).

In both tests, all the jobs were completed successfully without resubmission, and all the VMs were running without errors in the OpenNebula infrastructure. It is worth to mention that previous tests to this one were done until reach a production ready state. Some technical issues were faced. Additional configuration and tuning were done in different parts of the cloud, which initially were affecting performances and degrading services causing errors, mainly regarding some optional packages of OpenNebula installation: the improved *xmlparser* and the *nokogiri* for html parsing; but also other technical issues like the *sunstone* SQLite problems when used with NFS, and the reliability of a shared or dedicated node for the *sunstone* server. creation and deletion of the DIRAC Virtual Machine Instance.

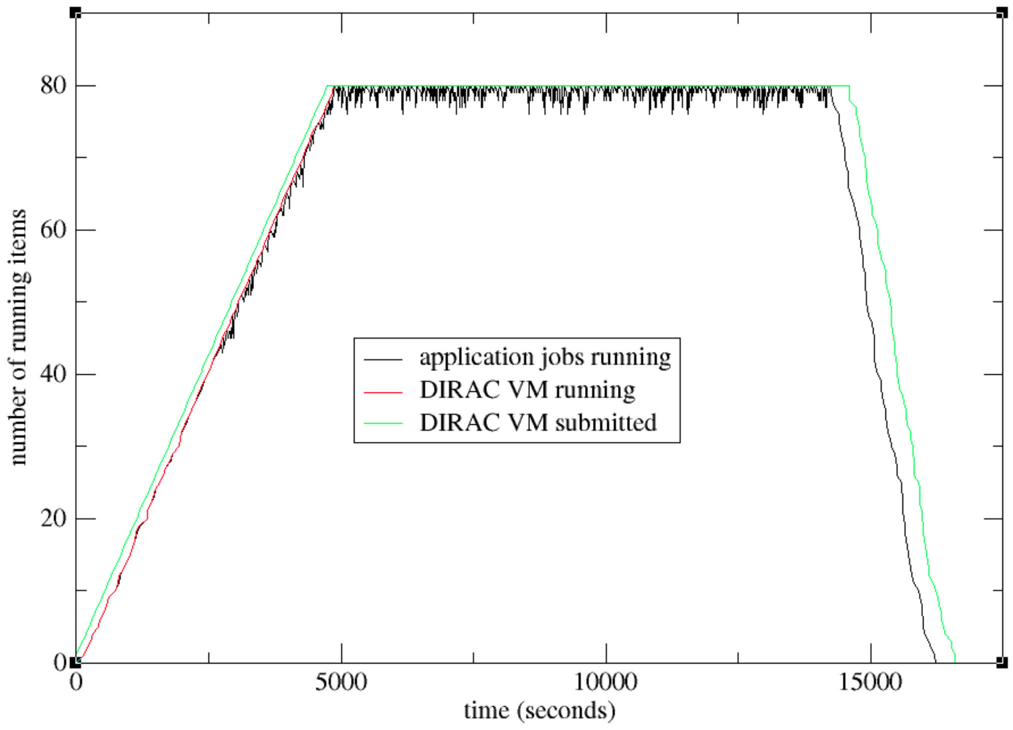

Figure 9. OCCI / OpenNebula Integration with DIRAC Test Results

The results of the test are shown in Fig. 9. The ramping-up shows the OCCI/OpenNebula submission and HEPIX contextualization overhead within green-red lines. On the same ramp-up within red-black lines one can see the DIRAC running VM overhead with the real jobs timestamps. On the plateau the VM submission line (green) and the VM running (red) are the same and equal to 80. At this point, all the VMs are submitted and running. Moreover, there is a DIRAC running VM overhead shown within green-black lines. The tail overhead in this test is about 15 min. for each VM and can be configured on the DIRAC Configuration Server.

Regarding the second metric of the test, aimed to obtain the KVM hypervisor overhead, the result in the Cloud testbed was 23 event/core/hour, while in the real Worker Node was 24,49 event/core/hour, which is about 6% of overhead in this worst scenario possible, with VMs of a single core.

7. Conclusions and Outlook

This paper describes the integration of the DIRAC software with different Cloud managers. The design of the new components has been detailed, proving the DIRAC software extensibility, based in open-source development standards and re-engineering software life cycle. This paper proves the successful coordination of different software appliances and middleware with DIRAC. Nowadays, this *interware* approach is beyond the frontline in computer science state of the art.

The described approaches make use of the flexibility provided by CVMFS and the HEPIX contextualization mechanism.

The success of the tests show different potential adoptions. Virtual overhead metrics are also showing the viability of the Cloud technologies for scientific computing. In this manner DIRAC is able to take advantage of the strong points of the Cloud technologies, regarding the platform maintenance, the dynamic environment composition of the computing resources and the hardware encapsulation on the computing nodes. The major Cloud weak, the performance overhead, has been measured, and it is shown that it is not an impediment for Cloud adoption in scientific computing.

Future work in DIRAC Cloud roadmap is including the VM multiplatform to cover different use cases requirements. Another work in progress is the design, implementation and deployment of a multi cloud endpoint management. The DIRAC integration with OCCI will be offered to the LHCb and Belle II experiments to deploy a production stage, initially with MC simulations, and then covering other aspects of their computing models: data processing and user analysis. Being a Core DIRAC functionality, once tested by these large collaborations these new development will be available to any community making use of DIRAC to build its distributed computing framework.

Acknowledgements
The Port dInformació Científica (PIC) is maintained through a collaboration between the Generalitat de Catalunya, CIEMAT, IFAE and the Universitat Autònoma de Barcelona. This work was supported in part by grants FPA2007-66152-C02-01/02 and FPA2010-21816-C02-01/02 from the Ministerio de Educación y Ciencia, Spain.

References

[1] Tsaregorodtsev A, Bargiotti M, Brook N, Casajus Ramo A, Castellani G, Charpentier P, Cioffi C, Closier J, Graciani Diaz R, Kuznetsov G, Li Y Y, Nandakumar R, Paterson S, Santinelli R, Smith A C, Miguelez M S and Jimenez S G 2008 *Journal of Physics: Conference Series* **119** 062048 URL `http://stacks.iop.org/1742-6596/119/i=6/a=062048`

[2] Graciani Diaz R, Casajus Ramo A, Carmona Agero A, Fifield T and Sevior M 2011 *Journal of Grid Computing* **9**(1) 65–79 ISSN 1570-7873 10.1007/s10723-010-9175-7 URL `http://dx.doi.org/10.1007/s10723-010-9175-7`

[3] CloudStack URL `http://www.cloud.com`

[4] OCCI-Working-Group URL `http://occi-wg.org`

[5] OpenNebula URL `http://opennebula.org/`

[6] OpenStack URL `http://openstack.org/`

[7] Jakob Blomer T F 2010 *Proceedings of 19th International Conference* pp 1–6

[8] Casajus A, Graciani R and the Lhcb Dirac Team 2010 *Journal of Physics: Conference Series* **219** 042033 URL `http://stacks.iop.org/1742-6596/219/i=4/a=042033`

[9] Víctor Fernández Albor José Saborido F G F and Cacheiro J L 2011 *IEEE Third International Conference on Cloud Computing Technology and Science (CLOUDCOM)* pp 537–541

[10] Pedrag Buncic C Aguado Sánchez J B, Harutyunyan A and Mudrinic M 2011 *The European Physical Journal Plus* **126** 1–8