PROOF on the Cloud for ALICE using PoD and OpenNebula

D Berzano[1,2], **S Bagnasco**[2], **R Brunetti**[2], **S Lusso**[2]

[1] Dip. di Fisica Sperimentale—Univ. di Torino—Via P. Giuria 1, 10125 Torino—Italy
[2] Istituto Nazionale di Fisica Nucleare—Via P. Giuria 1, 10125 Torino—Italy

E-mail: dario.berzano@to.infn.it

Abstract. In order to optimize the use and management of computing centres, their conversion to cloud facilities is becoming increasingly popular. In a medium to large cloud facility, many different virtual clusters may concur for the same resources: unused resources can be freed either by turning off idle virtual machines, or by lowering resources assigned to a virtual machine at runtime. PROOF, a ROOT-based *parallel* and *interactive* analysis framework, is officially endorsed in the computing model of the ALICE experiment as complementary to the Grid, and it has become very popular over the last three years. The locality of PROOF-based analysis facilities forces sysadmins to scavenge resources, yet the chaotic nature of user analysis tasks deems them unstable and inconstantly used, making PROOF a typical use-case for HPC cloud computing. Currently, PoD dynamically and easily provides a PROOF-enabled cluster by submitting agents to a job scheduler. Unfortunately, a Tier-2 does not comfortably share the same queue between interactive and batch jobs, due to the very large average time to completion of the latter: an elastic cloud approach would enable interactive virtual machines to temporarily subtract resources to the batch ones, without a noticeable impact on them. In this work we describe our setup of a dynamic PROOF-based cloud analysis facility based on PoD and OpenNebula, orchestrated by a simple and lightweight control daemon that makes virtualization transparent for the user.

1. PROOF-based analysis facilities and the ALICE Experiment

PROOF (Parallel ROOT facility) [1] is a framework that enables ROOT [2] to perform parallel and interactive data processing in "embarrassingly parallel" use cases, such as typical HEP event-based activities. Events are processed in parallel by independent ROOT sessions (the *workers*) on multiple machines; results are collected and merged afterwards by a *master* session.

PROOF enables computing resources to be used *interactively*: all the requested PROOF resources are ideally available almost immediately. Workers can also be added or removed at runtime, and in case a worker crashes the job would proceed on the others.

A conventional static PROOF installation requires the *xproofd* daemon to listen for incoming connections on one machine: upon connection it will communicate to static *xproofd* daemons on the other machines to provide the user with workers.

Since the time required to complete the analysis of a single event depends on the available computing power and the complexity of the event itself, events are not evenly distributed to the workers; instead, the workers ask for new events when they are finished analyzing. This pull architecture, called the *dynamic packetizer* [3], leads to a uniform completion time.

A typical use case for a PROOF-based analysis facility in HEP is the optimization of user analysis: interactivity enables the physicist to immediately re-run the analysis after each small change in the code and cut parameters.

Those analyses run usually for a few minutes; moreover, since the tasks are run *interactively* by the user, there is a demand for PROOF resources only during working hours. Such discontinuous occupancy means that local PROOF resources may be largely underused.

PROOF is also available on larger scale deployments, where analyses on bigger samples or even full simulations or reconstructions can be run interactively.

Such facilities are normally used by a larger number of users, and fast network connectivity attracts non-local users. However, having a single static PROOF master process accepting connections for all the users is an approach that does not scale very well and can undermine the stability of the whole system.

At this point it should be clear that static PROOF deployments pose some issues when resources are either underused or heavily loaded: a *dynamic* PROOF deployment would rather make unused resources available for some other task in the first case, and would absorb peak loads in the latter.

PROOF is officially part of the ALICE computing model since 2006, when the CAF (Central Analysis Facility) was first set up at CERN [4]. In 2010 a set of guidelines for AAFs (ALICE Analysis Facilities) [5] were adopted to provide a uniform working environment throughout all ALICE PROOF clusters. AAF specifications cover: authentication via Grid proxy, software management via AliEn Packman [6][7], access to local data via a distributed disk pool aggregated and served via Scalla/xrootd [8] where pool disks are installed on the PROOF servers themselves, dataset management and staging [9] and monitoring via web through MonALISA [10].

The virtual analysis facility we are going to discuss is compliant to most of the AAF guidelines.

2. The tools: PROOF on Demand and OpenNebula

2.1. PROOF on Demand

PROOF on Demand (PoD) [11] is a set of tools that enables PROOF to be instantiated dynamically on an existing batch farm, making no particular assumptions on the underlying batch system. There are plugins for a wide range of resource management systems. Another independent plugin, *pod-ssh*, does not even need a batch system and uses standard passwordless SSH to access the worker nodes.

Its components act as wrappers around PROOF; the most important are:

- *pod-server*—launched only once by the user, it sets up the PROOF master and monitors the status of the PROOF cluster; it also terminates the cluster automatically in case of inactivity;
- *pod-agent*—it prepares the environment for PROOF and controls a single PROOF worker throughout its whole lifecycle, firing it up and cleaning up temporary files in the end; it is meant to be deployed by a batch system via *pod-submit*;
- *pod-submit*—queues the specified number of *pod-agent* jobs to the batch system, requesting for a certain number of PROOF workers: each *pod-agent*, once deployed, will communicate with the corresponding *pod-server*.

As soon as the first worker becomes available the user can start using PROOF; further workers will join the facility as soon as they are ready, or can be added to it at a later time.

Since each user has a different PROOF master, PoD helps addressing the scalability issues of a large PROOF deployment.

2.2. OpenNebula

OpenNebula [12] is an open source IaaS (Infrastructure as a Service) framework to set up, administrate and monitor public and private clouds. OpenNebula takes care of the setup and provisioning of the virtual resources, including storage areas and network connectivity. All of these virtual resources, and the virtual machines themselves, are described inside specific *templates*.

We favored OpenNebula over similar tools because it is widely used and almost every functionality is implemented via Bash or Ruby scripts, making it easy to add missing functionalities or to tailor the existing ones around our use case. OpenNebula refers to these scripts as MADs (Middleware Access Drivers): there exist MADs for transferring images, monitoring, handling the authentication and supporting different virtualization methods.

As of v3.0, the supported hypervisors [13] are Xen [14], KVM [15] and VMWare [16], all of them implementing basic deployment functionalities such as suspension, migration and even live migration. An interface to Amazon EC2 [17] is also available, as well as OCCI APIs for public cloud setups. Different hypervisors or cloud interfaces can be used together to set up a hybrid cloud.

Virtual machine images are registered inside a non-hierarchical repository: the *transfer manager* MAD is responsible of transferring the images from the image repository to the hypervisor *(cloning)* and vice-versa *(saving)*. Transfer managers supplied by default with OpenNebula include: the NFS transfer manager, for images on a filesystem shared between all the hypervisors; the LVM transfer manager, to run VM images written on LVM partitions instead of plain image files; the SSH transfer manager, to transfer the images using *scp*.

OpenNebula also features a web interface called Sunstone [18] that supports custom plugins.

3. PROOF on Demand and OpenNebula interplay

A prototype of a private cloud based on OpenNebula is being set up in the computing centre of INFN-Torino, a WLCG Tier-2 centre. The cloud runs a dynamic PROOF cluster based on PoD that uses a dedicated batch system whose worker nodes are virtual machines created and destroyed dynamically using the mechanisms supplied by OpenNebula.

There are many advantages in using a batch system over using *pod-ssh*. First of all, *pod-ssh* connects to machines that are already up and running, while through *pod-submit* we could place the *pod-agents* in a queue and the batch system will deploy them in order when the machines are up.

Secondly, the batch system knows the structure of the cluster and maintains a register of what is currently running on it, making tasks wait in a queue until the resources to run them become available—*pod-ssh* does not feature any queue and would be likely to overload the nodes.

Furthermore, *pod-ssh* needs a list of the valid PoD-enabled machines to connect to: making this list dynamically provided and making pod-ssh wait until the machines are up would require the reimplementation of features that are core functionalities of any batch system.

A batch system is also useful to customize the jobs distribution and to enforce policies (while no particular limitation can be enforced via *pod-ssh*): for instance, we could configure the batch system to keep jobs of different users separated on different virtual machines, or limit the maximum number of concurrent *pod-agents* or the maximum walltime.

We decided to use TORQUE v2.4 [19], based on PBS and simple to configure, but any of the batch systems supported by PoD would in principle work fine.

3.1. Farm configuration

3.1.1. Configuration of the access node

In our setup the user submits the jobs via pod-submit to a single access node (in this case it acts as a wrapper around TORQUE's *qsub*) which runs the TORQUE server *(pbs_server)* and the job scheduler *(pbs_sched)*.

Given the very simple nature of our jobs, we were able to achieve an improved response time by configuring the schedule to reduce the default queue polling interval from 1 min to 7 s. This means that it takes TORQUE at most 7 s to dispatch the pod-agent to a freshly connected worker node.

Also, since *pod-agents* are disposable, we do not care to keep track of them in the queue when they are finished: the queue is configured to flush the agents as soon as they finish.

3.1.2. Configuration of the virtual dynamic worker nodes

Since worker nodes for PROOF are virtual machines, once the configuration is done on one machine the corresponding image can be cloned several times.

The virtual worker node runs the *pbs_mom* daemon that controls the running jobs and communicates with the server. Since the running worker nodes are created dynamically, the TORQUE server should be somehow made aware of the presence of the new node in order to start assigning jobs to it.

In our setup, virtual worker nodes add themselves automatically to the PoD queue by executing a command on the server via SSH right after the bootstrap: a private key is used for passwordless authentication, and for security reasons connections made with that key are limited to execute only one command, chosen to be the *qmgr* command that adds the node to the PoD queue.

This pull architecture is easier, less error-prone and quicker than adding nodes separately: in fact, if a node can successfully add itself to the queue it means that network connectivity works properly, and we avoid the cases where a virtual machine deployment fails, leaving a dangling node in "offline" state in the TORQUE nodes list.

Our operating system of choice for the virtual machines is CentOS 5.7, binary compatible with upstream RHEL releases, so that we can run precompiled ALICE software. The cloud environment currently runs on KVM hypervisors.

When running PROOF, the version of ROOT and AliRoot to use for the session is chosen upon connection: since this user decision happens way after virtual machine's boot, we cannot deploy a virtual machine self-containing only the requested software version; on the other side, a VM with all the ALICE software versions (tens of GB) would be both difficult to maintain and very clumsy to deploy.

ALICE software packages, as defined by AAF specifics, are exacty the same as the packages for the Grid worker nodes, so we just mount a shared ALICE Grid software area exported using the Lustre™ distributed filesystem [20], without the need for a software manager specific to PROOF.

In order to make the virtual machine image as small as possible, both the swap space and an empty scratch filesystem are created by OpenNebula upon the virtual machine deployment directly on the disks of the target hypervisor.

3.1.3. A physical PROOF cluster with a virtual elastic extension

It is worth noting that the configuration of the virtual worker nodes, apart from some configuration items specific to the virtualization (such as the dynamically-created disk partitions), is a perfectly valid configuration also for a physical machine. A virtual PROOF worker node and a physical one are, from the point of view of the batch system, nothing more than job slots, so we can actually use the batch system to deploy PROOF workers on a *physical* analysis facility with an *elastic extension* made out of virtual machines.

In this hybrid case the batch system can be, for instance, configured to favor physical over virtual worker nodes for performance reasons, a distinction that would not be possible to accomplish easily using *pod-ssh*.

3.1.4. Sharing virtual resources with other virtual clusters dynamically

The most important aspect of our PoD over OpenNebula cluster is the deployment swiftness: since interactive resources must be made available in a reasonably short amount of time, every configuration option has been tailored to gain precious seconds.

However, since we cannot afford to wait hours for some virtual resources to be freed by a certain virtual cluster, our PoD virtual machines can be configured to grab some CPU resources from other running virtual clusters without stopping them. This is currently supported on KVM by simply using *renice*, since virtual machines are treated as ordinary Unix processes.

Squeezing resources can not be done indiscriminately: for instance, it is agreed that an ALICE PROOF worker node can temporarily steal resources from a virtual Grid node running very long batch

jobs from the same VO. The static prototype of the Virtual Analysis Facility we set up in Torino in 2008 [21][22] is now much easier to set up as a special use case of our OpenNebula cloud.

3.2. The load sensor

Turning the PROOF virtual machines on can be done manually by the system administrator or by the end user, but in a production facility it is much better to automatize such tasks. Moreover, since PoD is a well-established tool and very easy to learn, we would like to make the user completely unaware of the underlying virtualization layer, by not changing the standard PoD workflow (*pod-server* followed by *pod-submit*) and without patching PoD code.

The batch system can keep track of *pod-agents* waiting in the queue: if there are any, actions can be taken accordingly, such as spawning new virtual machines. On the other side, the batch system monitors the status of the nodes, so that we can turn off idle nodes, *i.e.* nodes that are haven't been executing PROOF workers for a certain amount of time.

The load sensor is a daemon (called *bananad*) that performs the two tasks above. It continuously checks the queue for waiting jobs and the nodes status for idle nodes. It also removes stale "offline" nodes hanging in the batch system for too long.

Continuously polling the status of the queue via the *qstat* command is a resources-consuming task, so wherever possible TORQUE logs are opened and read in a separate thread as more lines are appended to them. This strategy makes the daemon very lightweight.

All the *bananad* thresholds are configurable: the idle time before a virtual node is turned off, how long an "offline" node is left there before being removed, and how long a job is left waiting before asking for new virtual machines.

It is very important to point out that *bananad* is the only piece of software having direct access to the cloud infrastructure, concealing it completely both to the end user and the batch system.

3.3. OpenNebula tweaks

OpenNebula is the tool of our choice because it is easily customizable through simple scripting. The following OpenNebula configuration has been obtained only by modifying such scripts.

3.3.1. Virtual bridges: network isolation

Our OpenNebula infrastructure is not PROOF-exclusive, but it runs a certain number of virtual clusters: different virtual machines of the same cluster running on different hypervisors appear as being attached to the same virtual network bridge.

This can be obtained by configuring an OpenNebula *hook* (an event callback) executed before starting the VM that uses *ebtables* [23], an iptables-like kernel space tool that operates at Ethernet level to isolate the class of incoming and outgoing MAC addresses to a virtual machine. The hook is executed on the hypervisor running the virtual machine, and it has been modified to allow the specification of a list of extra authorized MAC addresses directly inside the OpenNebula VM template. This is needed to allow all the VMs of the same cluster to communicate with a service that resides outside the virtual bridge.

3.3.2. Quick deployment of virtual machines on LVM

In our setup, virtual machines are run on LVM partitions for performance reasons. This increases the performances with respect to running them from image files.

The LVM Transfer Manager MAD provided by OpenNebula has been heavily modified to drastically improve the virtual machine deployment time.

In the default LVM module, the image is first copied as a file via *scp* on the hypervisor, then it is dumped on the LVM partition using *dd*. To avoid this double transfer, plus the connection overhead introduced by SSH, the image repository resides on a shared GlusterFS [24] filesystem mounted on all hypervisors, and the image is "disk dumped" directly to the LVM partition. With this method, the

Table 1. Time needed to create on the fly an ext3 and a XFS filesystem: the results show how a XFS filesystem of any size may be created in near-real-time. These results were obtained on non-SDD disks.

Size	ext3	XFS
20 GB	13.1 s	1.5 s
40 GB	20.0 s	1.6 s

transfer time of a 4 GB virtual machine has been reduced reduced from ~200 s of the standard LVM module (*scp* then *dd*) to the ~50 s of the modified LVM (direct *dd* over GlusterFS).

The second large improvement comes from the consideration that virtual machine images rarely change: it is therefore a waste of time to transfer a big image several times. Each virtual machine image is thus cached as a LVM partition, and the virtual machine instance is run on a snapshot partition of the image cache, using the *snapshot* feature built into LVM. Transferring the image to the hypervisor only happens if the image is not there yet, but further accesses only need to snapshot that partition, an operation that only takes a negligible amount of time (< 1 s).

Running VMs on snapshots of the original cache partition, without touching it, has some remarkable advantages. First of all, every potential damage made to a running virtual machine remains inside the snapshot and does not propagate to the original image. Moreover, multiple virtual machines can be run on the same hypervisor from multiple snapshots of the same LVM logical volume.

OpenNebula's LVM module also features functions to dynamically create disposable filesystems for the virtual machine. In particular, the swap creation module has been rewritten in order to create the swap partition on LVM too (OpenNebula's default module creates the swap on a file), and for the disposable scratch area XFS has been chosen because the filesystem creation is almost immediate with respect to some other well-established filesystems: as clearly shown in **table 1** the creation of a XFS filesystem does not depend much on its size, while the creation of an ext3 filesystem is slow (compared to our interactive use case) and strongly dependent on the filesystem size. All of our hypervisors use regular non-SDD SATA disks.

3.3.3. Easier updates through contextualization

Contextualization is the process of running a set of functions to customize a general-purpose virtual machine image at boot time. By using contextualization, instead of hardcoding some modifications inside the virtual machine, we can keep our image repository smaller and easier to update. In the PROOF case, for instance, since all the software is on a shared partition and not within the virtual machine, the instantiated virtual machine is just a basic CentOS 5 installation, and all the customization is made with a contextualization script that takes < 2 s to complete, thus not negatively affecting the boot time.

3.4. SSH authentication using Grid credentials and PoD connectivity

In this setup each user needs to login to the access node in order to run the PoD server: since v3.6 PoD features the *pod-remote* command, a local shell that issues remote PoD commands requiring only a SSH connection to the access node. The connection to PROOF is also tunneled through SSH, as depicted in **figure 1**, so that it is easy to configure a firewall allowing only *sshd* port (22/TCP) on the access node.

In order to allow ALICE users to transparently connect to SSH using their Grid credentials (a X.509 certificate and a private key), a two-step mechanism has been set up [25]. *Authentication* is performed via HTTPS using the certificate and the key: past this step, the user obtains a *temporary*

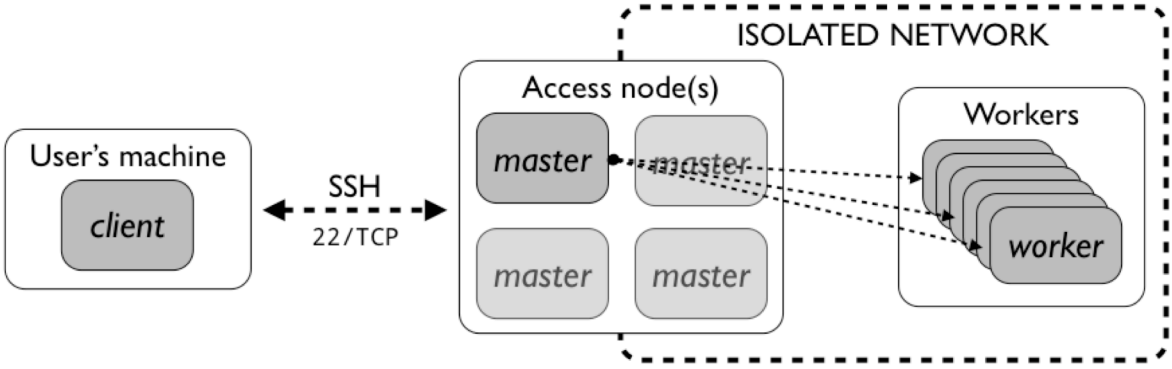

Figure 1. PoD connectivity schema: the only port opened for outbound connectivity is the SSH port, while both physical and virtual PROOF workers freely communicate in a private, isolated network. The access node, running the PROOF masters, acts as an interface between the two networks.

authorization to connect via SSH with private key authentication by presenting his or her Grid private key, which happens to be in the same format (PEM) recognized by SSH.

Certificate to Unix user mapping is performed using the ALICE LDAP user database, and home directories are created on the fly using the PAM *mkhomedir* module.

This approach has been made transparent to the user, which is only prompted for a passphrase to unlock the Grid private key.

4. Conclusions

This work shows how, using mainstream and widespread tools, it is possible to build a virtual elastic interactive analysis facility with a wise configuration of the single tools and minimal development effort.

Every piece has been configured by keeping in mind that interactive resources must be made available as soon as possible: the batch system has been configured accordingly, while the cloud infrastructure has been tweaked to avoid useless time-consuming transfers, through a fast shared filesystem and local caching of the VM images. Once all the images have been cached, the maximum time a user waits to obtain 40 PROOF workers, even when the resources must be shared with other running virtual clusters, is under 1 min 30 s, perfectly acceptable for an interactive resource.

With this work we have shown that it is feasible to use an IaaS tool like OpenNebula to dynamically provide for interactive PROOF resources even on a small-sized computing facility such as a Grid Tier-2 centre.

References

[1] Ballintijn M, Brun R, Rademakers F and Roland G 2003 *Arxiv preprint physics/0306110* (see also http://root.cern.ch/drupal/content/proof)
[2] Brun R and Rademakers F 1997 *Nucl. Instr. and Meth. in Phys. Res.* A **389** 81 (see also http://root.cern.ch/)
[3] Ganis G, Iwaszkiewicz J and Rademakers F 2007 *Proceedings of Science* PoS(ACAT)022
[4] Grosse-Oetringhaus J F 2008 *J. Phys.: Conf. Ser.* **119** 072017
[5] http://aaf.cern.ch/
[6] Bagnasco S, Betev L, Buncic P, Carminati F, Cirstoiu C, Grigoras C, Hayrapetyan A, Harutyunyan A, Peters A J and P Saiz 2008 *J. Phys.: Conf. Series* **119** 062012
[7] http://alien2.cern.ch/index.php?option=com_content&view=article&id=39&Itemid=86
[8] Furano F 2011 *Eur. Phys. J. Plus* **126** 12

[9] http://indico.cern.ch/getFile.py/access?contribId=66&sessionId=6&resId=0&materialId=slides&confId=138471
[10] Legrand I, Voicu R, Cirstoiu C, Grigoras C, Betev L and Costan A 2009 *Comm. of the ACM* **52** (9) 49
[11] Malzacher P and Manafov A 2010 *J. Phys.: Conf. Ser.* **219** 072009 (see also http://pod.gsi.de/)
[12] http://opennebula.org/
[13] http://opennebula.org/documentation:rel3.0:vmmg
[14] http://xen.org/
[15] http://www.linux-kvm.org/page/Main_Page
[16] http://www.vmware.com/
[17] http://aws.amazon.com/ec2/
[18] http://opennebula.org/documentation:rel3.0:sunstone
[19] http://www.adaptivecomputing.com/products/torque.php
[20] http://www.lustre.org/
[21] Berzano D, Bagnasco S, Lusso S and Masera M 2008 *Proceedings of Science* PoS(ACAT08)050
[22] Bagnasco S, Berzano D, Lusso S, Masera M 2009 *J. Phys.: Conf. Ser.* **219** 062033
[23] http://ebtables.sourceforge.net/
[24] http://www.gluster.org/
[25] Berzano D 2011 *INFN-CCR internal note* (submitted) CCR-42/2011/P