IOPscience iopscience.iop.org

Home Search Collections Journals About Contact us My IOPscience

Lightweight scheduling of elastic analysis containers in a competitive cloud environment: a Docked Analysis Facility for ALICE

This content has been downloaded from IOPscience. Please scroll down to see the full text.

2015 J. Phys.: Conf. Ser. 664 022005

(http://iopscience.iop.org/1742-6596/664/2/022005)

View the table of contents for this issue, or go to the journal homepage for more

Download details:

IP Address: 188.184.3.52
This content was downloaded on 06/01/2016 at 16:05

Please note that terms and conditions apply.

Journal of Physics: Conference Series **664** (2015) 022005 doi:10.1088/1742-6596/664/2/022005

Lightweight scheduling of elastic analysis containers in a competitive cloud environment: a Docked Analysis Facility for ALICE

D Berzano[1], J Blomer, P Buncic[1], I Charalampidis, G Ganis and R Meusel

European Organization for Nuclear Research (CERN), Genève, Switzerland

[1] For the ALICE Collaboration

E-mail: dario.berzano@cern.ch

Abstract. During the last years, several Grid computing centres chose virtualization as a better way to manage diverse use cases with self-consistent environments on the same bare infrastructure. The maturity of control interfaces (such as OpenNebula and OpenStack) opened the possibility to easily change the amount of resources assigned to each use case by simply turning on and off virtual machines. Some of those private clouds use, in production, copies of the Virtual Analysis Facility, a fully virtualized and self-contained batch analysis cluster capable of expanding and shrinking automatically upon need: however, resources starvation occurs frequently as expansion has to compete with other virtual machines running long-living batch jobs. Such batch nodes cannot relinquish their resources in a timely fashion: the more jobs they run, the longer it takes to drain them and shut off, and making one-job virtual machines introduces a non-negligible virtualization overhead. By improving several components of the Virtual Analysis Facility we have realized an experimental "Docked" Analysis Facility for ALICE, which leverages containers instead of virtual machines for providing performance and security isolation. We will present the techniques we have used to address practical problems, such as software provisioning through CVMFS, as well as our considerations on the maturity of containers for High Performance Computing. As the abstraction layer is thinner, our Docked Analysis Facilities may feature a more fine-grained sizing, down to single-job node containers: we will show how this approach will positively impact automatic cluster resizing by deploying lightweight pilot containers instead of replacing central queue polls.

1 Background

Cloud and virtualization technologies have always generated a relevant interest in High Energy Physics computing, largely dominated by the Grid. Local Grid deployments running virtualized on top of private clouds such as OpenNebula [1] and OpenStack [2] are nowadays mature, as they allow more degrees of freedom on the physical infrastructure configuration while ensuring a consistent environment for Grid jobs. Among the advantages, we mention the ability to choose any operating system for the underlying hypervisors, and the flexibility to expand and compress Grid resources to make room for spot use cases without requiring a separate dedicated infrastructure. Virtualization has also a positive impact on high availability: important services can be proactively moved from faulty to healthy servers without interruption (live migration), and outdated worker nodes can be gradually and dynamically replaced with freshly updated ones as soon as they phase out (rolling updates).

Even if cloud computing has also opened the way for direct virtual machines submission in place of

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

Grid jobs, the success of virtualization in HEP is due to its transparency: users continue to submit ordinary Grid jobs using standard interfaces and they are unaware of the nature of the exploited resources. Small to large sites have adopted this approach: for instance, the ALICE Tier-2 in Torino, based on OpenNebula [3], and the CERN site, where worker nodes are virtual machines managed by a single OpenStack deployment [4] that transparently aggregates the CERN computing center located in Switzerland and the Wigner Data Center in Hungary.

1.1 The Virtual Analysis Facility and its ALICE applications
The Virtual Analysis Facility [3][5], also known as CernVM Elastic Clusters, is an implementation of a portable preconfigured batch cluster that addresses the problems of usability, scalability and fair resources exploitation.

An Elastic Cluster is constituted by a head node and a variable number of workers. The head node is used to submit and control batch jobs, and the number of worker nodes varies dynamically based on the effective usage. The batch system in use is HTCondor [6], as it supports dynamic reconfiguration of workers; elastiq [7] is used to monitor the status of HTCondor to start new virtual machines when too many jobs are waiting, and turn them off when they turn idle.

Every tool, including elastiq, is embedded in the virtual cluster: the end user simply deploys a single virtual machine (the head node), and worker nodes will be launched transparently only when needed, without further direct user intervention.
Cloud interaction occurs using the standard EC2 API [8], supported by all major cloud orchestrators: the Elastic Cluster is a “batch system in a box” that can be deployed and scaled without any deep knowledge of either batch systems or cloud computing.

The ALICE experiment is currently using different instances of the Virtual Analysis Facility in a number of critical use cases.

1.1.1 Opportunistic cloud on the High Level Trigger. This cluster [10] counts the equivalent of 7000 Grid job slots (with hyperthreading enabled), and more than 3 GB of RAM per slot. The High Level Trigger is a mission critical computing facility used during data taking: for this reason it is isolated from the external network. Outside data taking we exploit unused resources by running an OpenStack cloud on top of it, where an Elastic Cluster is automatically deployed by means of elastiq.

Virtual machines provide us with a thick layer of isolation with the High Level Trigger’s specific configuration, and its administrators can select which nodes are part of the cloud: when a new hypervisor is activated, elastiq reacts by filling it with virtual machines without manual interventions. This deployment has been tested successfully and has been commissioned in May 2015.

1.1.2 Disposable Release Validation Cluster. This is an Elastic Cluster running long batch validation tests on ALICE software release candidates [11]. The Release Validation Cluster is fully disposable: validation results are saved on a web-browsable shared storage, and the cluster itself can be completely destroyed after use. The cluster can be run anywhere, but for centrally managed operations the CERN Agile Infrastructure [4] is used.

1.1.3 PROOF-based Virtual Analysis Facility. This is the first Elastic Cluster application ever used by ALICE [5]: HTCondor batch resources are used interactively by means of PROOF on Demand [12]. The first of this kind has been in production in Torino since early 2012 [13], and has been subsequently evolved in the generic CernVM Elastic Clusters implementation.

1.2 Issues of cloud deployments
The reasons why we use virtual machines can be summarized in the ability of running on the same hardware several non-interfering use cases *(isolation)* exposing a known runtime environment to their applications *(consistency)* by adapting the amount of dedicated resources at runtime *(elasticity)*.

The Grid is the major stakeholder in High Energy Physics: by construction, it tends to saturate all

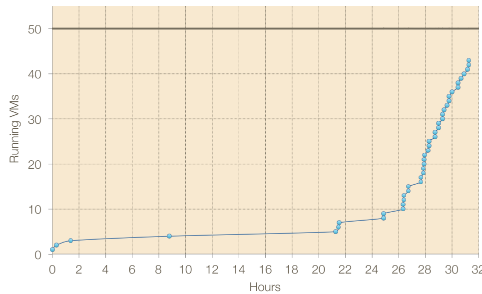

Figure 1. Observation of potential inelasticity on CERN OpenStack: out of 50 VMs requested only 10 VMs were up after 24 hours.

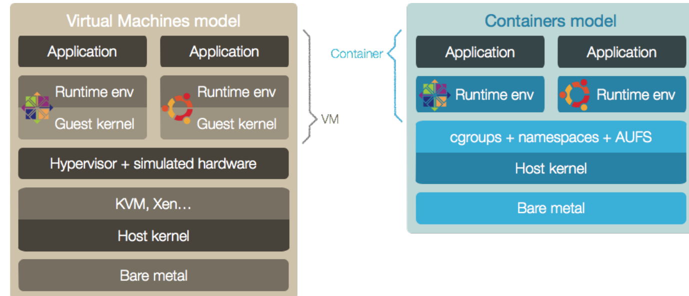

Figure 2. Containers and VMs compared: applications run on the bare metal in containers but the guest kernel cannot be changed.

the available resources on a private cloud, meaning that all smaller elastic appliances might not find enough resources to scale up: an example is shown in Figure 1.

It is still possible to use the available resources at their maximum by making the largest use cases periodically relinquish their quotas in order to allow smaller applications to fill in. This sort of preemption encounters however the same problems we would face when performing rolling updates: existing virtual machines must be set to *drain mode*, meaning that they do not accept new jobs but they let only the existing ones to finish before terminating the VM.

ALICE jobs however have a running time up to 48 hours, meaning that, in the worst case, we will have a single job slot running and all the other idle for two days before we can effectively use them. Backfilling techniques are of course possible: free slots can be filled with a series of shorter jobs, however we might simply not have enough short jobs to insert. Via backfilling, for what concerns ALICE and in general all HEP experiments, the problem can be mitigated but not completely solved.

Ideally, scheduling issues can be partially solved by making virtual machines as small as possible, down to one-job nodes: a virtual machine would live for the time of its only job, and no draining (and no job slots wasted) would occur. There is, however, a tradeoff between reducing the overhead introduced by a VM (a virtual operating system, virtual hardware allocated, etc.) and a more efficient scheduling: smaller VMs have a larger relative overhead per job slot, which inevitably leads to reduced performances.

In the next paragraphs we will show how adopting solutions based on technologies other than virtualization might lead to a more efficient scheduling *and* the same level of consistency and isolation provided by virtual machines.

2 Containers

In recent times, containers have become an interesting alternative to virtual machines for what concerns our intended use cases: recent Linux kernels have introduced a series of features that provide the desired level of isolation without introducing the overhead required to run a whole virtualized environment.

From a user application perspective, a container has essentially two levels of isolation combined: *cgroups* [14] is a Linux kernel facility that allows to account and cap the amount of resources (most notably resident memory, swap and CPU time) used by a container; *kernel namespaces* [15] limit the visibility scope of network interfaces, processes, users and more. Limiting the visibility of mount points and filesystem hierarchy is also done through namespaces, representing an evolution of the legacy Unix *chroot* facility.

Even if containers are often referred to as "lightweight virtual machines", they cannot be fully compared to them. Virtual machines are like ordinary computers that interact with virtual hardware exposed by their hosting hypervisor: a virtual machine has a virtual BIOS and a virtual RAM memory, and it is likely to have a virtual network interface and a virtual block device—all of them treated as if

they were real ones by the virtual machine's operating system (note that this simplification does not take into account *paravirtualized* hardware drivers).

Applications running inside a virtual machine also benefit from a fully emulated operating system: this includes a different kernel from the running hypervisor, in addition to a different runtime environment. Virtual hardware and the guest kernel are additional layers that separate the application from the bare metal.

Container applications, on the other hand, run directly on the kernel of the hosting machine: the aforementioned isolation features allow exposing a chrooted guest runtime environment different from the host's, but the operating system's kernel cannot be changed. A schematic comparison between virtual machines and containers is presented in **Error! Reference source not found.**.

2.1 Linux Containers and Docker

In the last years several low level technologies have been developed to implement isolation at kernel level: other technologies have been built on top in order to make them effectively usable for creating and containers and easily deploying applications with them.

Linux Containers (LXC) [16] combine cgroups and kernel namespaces to provide a sandboxed environment for applications. Docker [17], which primarily uses LXC as virtualization engine, provides support for storing, versioning and deploying base containers.

In Docker, every base image is a read-only root filesystem hierarchy. A running container can be created out of it: in this case, the read-only part is overlaid with a read-write filesystem where only the modifications will be saved. AUFS [18] or OverlayFS [19] are used for that purpose. Docker also supports block device overlays using device-mapper's thin provisioning (dm-thinp) [20].

Every writable layer contains only the modifications to the base layer, and can be saved in turn to become a new base image: each new container is therefore the result of overlaying a number of read-only filesystems, plus a single read-write filesystem on top.

The idea of versioning is heavily inspired by Git. Docker also easily allows to download ("pull") official base images and user contributions from a central repository [22] or custom ones, as well as uploading ("push") your own. Each layer is identified by a unique hash. Similarly to Git, small modifications or branching with respect to a larger base image are very lightweight to upload and deploy, as only the difference and a pointer to the previous layer are saved.

2.2 From virtual clusters to container-based clusters

Containers have some limitations compared to virtual machines: for instance, the kernel cannot be customized and containers cannot be currently migrated. However, our current Virtual Analysis Facility applications require a level of isolation that make Docker containers a suitable alternative to virtual machines for their deployment.

Since ALICE software is distributed with CernVM-FS, we need containers to support it; moreover, we would like to use the CernVM environment as a Docker container instead of as a virtual machine. The most relevant feature of CernVM is that its filesystem comes from CernVM-FS and it is downloaded on demand: we would like CernVM-based Docker containers to retain this feature. In section 3 we illustrate how we have managed to use the CernVM environment and CernVM-FS repositories with Docker.

Containers bring little overhead to our applications, compared to virtual machines, making it possible to encapsulate single job inside a container. Instead of mimicking virtual machine deployment models, we have addressed scheduling and scalability issues by considering that we need to deploy more containers than virtual machines, and more frequently. A pilot container and factory model and possible implementations are discussed in section 4.

3 CernVM and CVMFS in a Docker container

CernVM-FS is a mounted filesystem needing the FUSE kernel module in order to work. By default, Linux Containers run in *unprivileged* mode, where kernel cannot be tampered with, and root user

privileges are limited to the scope of the container: in order to mount a CernVM-FS repository, we would need to run containers in *privileged* mode, losing all the isolation and security layers.

Both for the case of running CernVM and accessing CernVM-FS repositories we have used solutions that expose a CernVM-FS filesystem mounted on the host to the containers

3.1 Native CernVM image as a Docker container

We have seen how Docker overlays read-only and read-write filesystems to create the container filesystem. Incidentally, this is the same technique used by CernVM: starting from CernVM 3 [23], the root filesystem comes from CernVM-FS (which is read-only), and it appears writable thanks to an overlay with AUFS [18].

We have realized a demonstrator called *docker-cernvm* [24] that leverages such analogy to register the CernVM read-only filesystem as a Docker container: as a result, the same unmodified CernVM environment used for the virtual machine image can be run as a normal container by means of *docker run*. A writable layer is created by Docker on top of it, as it normally does for any base container.

The *docker-cernvm* script automatizes two processes: registering a dummy image to the Docker repository and mounting the actual CernVM filesystem on top of it.

To register a dummy image, the following command is executed:

docker-cernvm --tag alice/cernvm register

A Docker image called *alice/cernvm* will be created: the image contains a single placeholder file used by the mount operation to identify the mountpoint. Registration is done once without root privileges. The next operation mounts CernVM in place of the dummy image:

docker-cernvm --tag alice/cernvm mount

Mounting requires root privileges. Once completed, the container can be run as any other container.

The demonstrator shows that it is natural to run CernVM as a Docker container: as a further example, the pilot container prototype presented in section 4.1 is based on CernVM.

It is interesting that, thanks to CernVM-FS, the CernVM base image does not have to make compromises with respect to other “minimalistic” or “core” operating systems in order to be lightweight to deploy: CernVM is fully fledged, and only the needed parts are fetched on demand.

3.2 CernVM-FS access from a container

The easiest and most efficient way to use CernVM-FS repositories inside a Docker container is by using the Docker “volumes” feature, accessible with the *-v* switch of the *docker run* commands: CernVM-FS repositories are mounted on the host machine, and exposed to containers via bind mounts in their filesystem hierarchy to the expected location.

The approach is has the clear advantage of preserving and sharing the CernVM-FS cache on the host machine, instead of losing it with each container. This means that every new container will benefit from content cached by previous container runs, and using disposable one job containers becomes sustainable.

Exposing CernVM-FS mount points to containers is straightforward, but the procedure has some *caveats*. On standard setups, CernVM-FS is configured with autofs to mount repositories the first time they are accessed: this does not play well with Docker volumes, and requires either that the repository is accessed right before starting the container (for instance through a dummy *stat* or *ls* operation), or that autofs is turned off completely and selected repositories are mounted explicitly. The latter is preferred for security reasons, and it can be automatized by adding CernVM-FS entries in */etc/fstab*.

In cases where the host cannot be configured for using CernVM-FS or it is not accessible, Parrot provides a viable solution to access CernVM-FS repositories. Even in such a case, CernVM-FS can be configured to use an external (“alien”) cache shared between containers.

4 Pilot containers: a "docked" analysis facility

As we have seen so far, containers are a radically different tool compared to virtual machines, making new solutions possible for deploying them on the large scale.

A possible container-based "docked" analysis facility should be capable of running the same jobs supported by the Virtual Analysis Facility. Since the VAF is, quite simply, a virtual HTCondor elastic cluster, a "Docked" Analysis Facility should just be capable of running HTCondor jobs as well.

The HTCondor community is currently working on a "Docker universe": it is still under development and it should make HTCondor directly capable of wrapping jobs within Docker containers [25].

We have preferred a different approach, where HTCondor runs inside specially configured containers called *pilot containers* (section 4.1) deployed by several *container factories* (section 4.2) not necessarily deploying HTCondor pilots: this extends the scope of the current Virtual Analysis Facility beyond batch jobs.

The most common problem of overlaying two schedulers (the one deploying virtual machines and the one dispatching jobs) is the lack of synchronization between them, often enforced by the separation of administrative domains. In a competitive cloud environment exploited by several "elastic" tenants this has a consequence: no tenant is willing to relinquish its resources, as nobody guarantees that they will be given back promptly, as clearly represented in **Error! Reference source not found.**. External solutions must be put in place, such as defining a maximum lifetime for VMs, in order to work around it.

The pilot approach proposed here works by eliminating scheduling of containers completely: they are constantly executed and exit immediately if there is nothing to do. The only scheduling performed occurs for the applications running inside containers.

In the next paragraphs we will see how we tackle this approach, possibly by integrating it with some existing solutions that currently target virtual machines, most notably Vac [26][27].

4.1 Anatomy of a HTCondor pilot container

A HTCondor pilot container is a Docker container behaving as a specially configured HTCondor execute node: it only runs the "master" and the "startd" daemons.

The container has a single startup script that starts the HTCondor daemons. HTCondor then connects back to a central "collector": this is done by means of the Condor Connection Broker (CCB) [28], requiring no open ports on the container side, that works even behind a firewall.

After an initial waiting time, the startup script on the container issues the *condor_off -peaceful* command: if the container is running a job, it will terminate as soon as the job is done. If the container has not received any job, it exits immediately. This configuration is very similar to glideinWMS [29] that creates a HTCondor cluster on top of any batch system. It has been shown that a pilot-based approach to HTCondor can scale up to O(100 000) concurrent jobs [30].

This configuration does not require anything but Docker installed on the host system, as HTCondor and its configuration are embedded in the pilot container. Being self-contained, a single pilot container setup is suitable to run both on dedicated and opportunistic facilities, and can be easily used for volunteer computing as well.

The startup time of pilot containers has been compared with the startup time of an analog pilot virtual machine, both based on the same CernVM version: by taking into account deployment, boot and registration to the HTCondor collector, a set of 48 distinct one-core pilot containers are ready to execute jobs in approximately 15 seconds, whereas 12 virtual machines with 4 cores each take almost 6 minutes on the CERN Agile Infrastructure. The huge difference is due to a number of factors: containers do not need to be "deployed" (Docker just creates an overlay), whereas virtual machines do; moreover, a pilot container only runs HTCondor, whereas a virtual machine has a complex boot process starting potentially unneeded services.

From the numbers it is clear that containers are more suitable than virtual machines for implementing a pilot model.

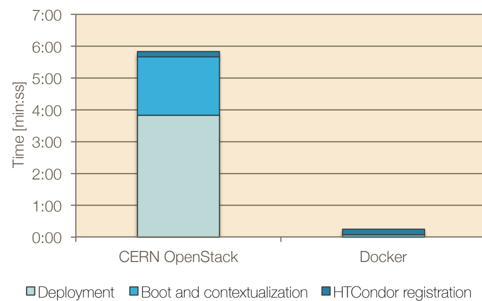

Figure 3. Average startup time of 48 HTCondor job slots (12 virtual machines with 4 cores each vs. 48 single-core containers). Both setups use the same CernVM image. Containers join the cluster within 15 seconds on average, where virtual machines have an overhead that pushes this time to almost 6 minutes on the CERN Agile Infrastructure.

In addition, it is worth noting how a pilot container can cleanly exit by itself: virtual machines, in some of the most common cloud deployments (including OpenNebula and OpenStack), cannot cleanly shutdown themselves, but they need to make an API call to request their clean termination.

An implementation of the pilot startup script working with the CernVM Docker container (section 3.1) is available for download [24].

4.2 The container factory

Following the concept of scheduling jobs, not containers, the best way of deploying pilot containers is to use a distributed set of *pilot factories*, without a single central control point. This approach tackles both scalability (even, possibly, on the geographical scale or opportunistic resources) and ease of configuration, as new or custom container factories can be dynamically added and removed.

Distributed pilot factories can run on each host: they can be configured to produce containers of different types (for instance, an Ubuntu or a CernVM container). Containers are expected to exit autonomously either when they have nothing to do or right after finishing their only job (section 4.1), however a container factory should kill containers running for too long.

Whenever a container of a certain type exits, the container factory immediately creates a new container. Container types can be selected in a round-robin fashion or randomly: in the latter case, relative probabilities can be configured to make some container types more likely to be produced than others. Probabilities can also be adjusted dynamically by lowering chances of producing containers that are more likely to terminate without doing anything.

Vac [26][27] implements an analog approach using distributed factories of pilot virtual machines: a collaboration with the Vac development team is envisioned in order to extend its scope to Docker containers.

5 Conclusions and future plans

Using the *docker-cernvm* demonstrator [24] we have shown that it is natural to run an unmodified CernVM as a Docker container by naturally exploiting overlay concepts shared by both systems.

CernVM-FS has versioning features as well, which the CernVM filesystem leverages to transitionally snapshot its upgrades. It is currently in the plans to support mounting multiple snapshots of the same repository: this feature is required to make them directly accessible from the Docker images repository. This feature will allow creating CernVM-based appliances where only a thin differences layer is distributed and overlaid on top of a specific CernVM snapshot, unleashing the full potential of Docker and CernVM combined.

A simple yet fully functional pilot container appliance based on HTCondor has been developed, showing that the time necessary to launch and boot it is much less compared to an analog virtual machine: containers are clearly more appropriate than VMs for a pilot-based pull scheduling model.

A comprehensible set of requirements for distributed container factories has been defined: most of them appear to be satisfied by Vac, a tool originally developed for running pilot virtual machines. We are exploring the possibility to collaborate with the Vac team to extend its scope to Docker containers.

Finally, it is worth mentioning Apache Mesos [31] as another popular containers deployment tool. Mesos covers a broader use case and it is commonly not used to run batch jobs: it is in the plans to verify how Apache Mesos behaves when running run HTCondor-based pilot containers, which would provide a compatibility layer needed to run our Virtual Analysis Facility applications.

References

[1] http://opennebula.org/
[2] https://www.openstack.org/
[3] D Berzano 2014 A ground-up approach to High-Throughput Cloud Computing in High-Energy Physics *Ph.D. Thesis* Università degli Studi di Torino, Italy
[4] RM Llamas *et al.* 2014 Commissioning the CERN IT Agile Infrastructure with experiment workloads *J. Phys.: Conf. Ser.* **513** 032066 doi:10.1088/1742-6596/513/3/032066
[5] D Berzano *et al.* 2014 PROOF as a Service on the Cloud: a Virtual Analysis Facility based on the CernVM ecosystem *J. Phys.: Conf. Ser.* **513** 032007 doi:10.1088/1742-6596/513/3/032007
[6] http://research.cs.wisc.edu/htcondor/
[7] https://github.com/dberzano/elastiq
[8] http://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html
[9] The ALICE Collaboration 2008 The ALICE experiment at the CERN LHC *Journal of Instrumentation* **3** S08002 doi:10.1088/1748-0221/3/08/S08002
[10] M Krzewicki, D Berzano *et al.* 2015 The ALICE High Level Trigger, status and plans *(submitted)* https://indico.cern.ch/event/304944/session/1/contribution/502
[11] D Berzano and M Krzewicki 2015 The ALICE Software Release Validation cluster *(submitted)* http://indico.cern.ch/event/304944/session/10/contribution/460
[12] P Malzacher and A Manafov 2010 PROOF on Demand *J. Phys.: Conf. Ser.* **219** 072009 doi:10.1088/1742-6596/219/7/072009
[13] D Berzano *et al.* 2012 PROOF on the Cloud for ALICE using PoD and OpenNebula *J. Phys.: Conf. Ser.* **368** 012019 doi:10.1088/1742-6596/368/1/012019
[14] https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt
[15] https://lwn.net/Articles/531114/
[16] https://linuxcontainers.org/
[17] https://www.docker.com/
[18] http://aufs.sourceforge.net/
[19] http://www.phoronix.com/scan.php?page=news_item&px=MTc5OTc
[20] https://www.kernel.org/doc/Documentation/device-mapper/thin-provisioning.txt
[21] http://ccl.cse.nd.edu/software/manuals/parrot.html
[22] https://hub.docker.com/
[23] J Blomer *et al.* 2014 Micro-CernVM: slashing the cost of building and deploying virtual machines *J. Phys.: Conf. Ser.* **513** 032009 doi:10.1088/1742-6596/513/3/032009
[24] https://github.com/dberzano/cernvm-alice-docker
[25] http://research.cs.wisc.edu/htcondor/manual/latest/2_12Docker_Universe.html
[26] A McNab *et al.* 2014 Running jobs in the vacuum *J. Phys.: Conf. Ser.* **513** 032065 doi:10.1088/1742-6596/513/3/032065
[27] http://www.gridpp.ac.uk/vac/
[28] http://www.slideserve.com/rene/ccb-the-condor-connection-broker
[29] I Sfiligoi 2008 glideinWMS—a generic pilot-based workload management system *J. Phys.: Conf. Ser.* **119** 062044 doi:10.1088/1742-6596/119/6/062044
[30] D Bradley *et al.* 2011 An update on the scalability limits of the Condor batch system *J. Phys.: Conf. Ser.* **331** 062002 doi:10.1088/1742-6596/331/6/062002
[31] http://mesos.apache.org/