Journal of Physics: Conference Series **219** (2010) 052010 doi:10.1088/1742-6596/219/5/052010

Integration of Virtualized Worker Nodes in Standard Batch Systems

Volker Büge[1], Hermann Hessling[3], Yves Kemp[2], Marcel Kunze[1], Oliver Oberst[1], Günter Quast[1], Armin Scheurer[1] and Owen Synge[2]

[1] Karlsruhe Institute of Technology , Postfach 6980, 76128 Karlsruhe, Germany
[2] DESY, Notkestraße 85, 22607 Hamburg Germany
[3] HTW Berlin, University of Applied Sciences, 10318 Berlin, Germany

E-mail: Oliver.Oberst@iwr.fzk.de

Abstract. Current experiments in HEP only use a limited number of operating system flavours. Their software might only be validated on one single OS platform. Resource providers might have other operating systems of choice for the installation of the batch infrastructure. This is especially the case if a cluster is shared with other communities, or communities that have stricter security requirements. One solution would be to statically divide the cluster into separated sub-clusters. In such a scenario, no opportunistic distribution of the load can be achieved, resulting in a poor overall utilization efficiency. Another approach is to make the batch system aware of virtualization, and to provide each community with its favoured operating system in a virtual machine. Here, the scheduler has full flexibility, resulting in a better overall efficiency of the resources. In our contribution, we present a lightweight concept for the integration of virtual worker nodes into standard batch systems. The virtual machines are started on the worker nodes just before jobs are executed there. No meta–scheduling is introduced. We demonstrate two prototype implementations, one based on the Sun Grid Engine (SGE), the other using Maui/Torque as a batch system. Both solutions support local job as well as Grid job submission. The hypervisors currently used are Xen and KVM, a port to another system is easily envisageable. To better handle different virtual machines on the physical host, the management solution VmImageManager is developed. We will present first experience from running the two prototype implementations. In a last part, we will show the potential future use of this lightweight concept when integrated into high-level (i.e. Grid) work-flows.

1. Introduction

There are at least two different reasons why one should virtualize the worker nodes of a High Throughput Computing (HTC) cluster. Either, in the site perspective, one has to run on a system with a setup which does not cope the needs of the users in security, privacy and compatibility aspects or, in the Grid perspective, one wants to provide the Grid users the possibility to choose between different environments for their jobs. In both cases, one wants to assure the simplest and most transparent approach possible to ease maintenance and management of the whole system. The virtualization layer should be completely hidden from the users, so that they only have to choose which environment they want to be served with. The virtualization of the worker nodes in fact results in a dynamic partitioning of a cluster as seen in figure 1. All concurrently running virtual machines of the same type hereby represent a partition of the cluster. In contrast to a static partitioned cluster, these dynamic partitions are

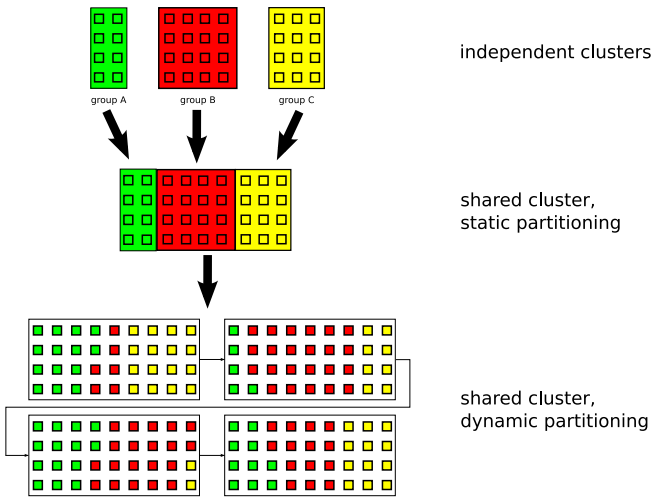

Figure 1. Three typical possibilities to run a HTC infrastructure. The first scenario shows independent clusters maintained by the specific user groups. The second and third scenarios show shared, centralised clusters which run on the same hardware infrastructure. In these cases it can either be offered with statically or dynamically partitioned sub-clusters.

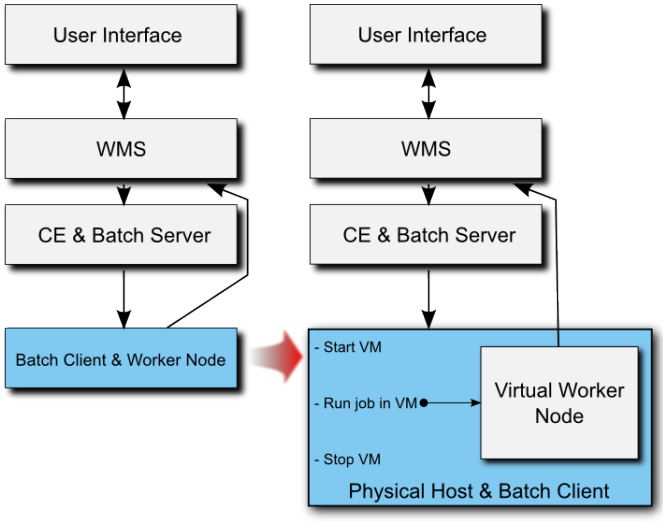

Figure 2. Grid job workflows. In contrast of having the worker node as a batch client on the hardware, the virtualized worker node which runs the job of the user is decoupled from the batch system. The Batch client on the hardware starts the VM via prologue and epilogue scripts and passes the job of the user to the VM. In fact the batch server considers the VM as part of the Job.

able to intercept idle times and peak demands of the different user groups and therefore assure an opportunistic distribution of the load on the resource, optimising the overall utilization efficiency of the cluster.

Earlier work on the dynamic partitioning of a shared computing cluster, has been deployed at the Karlsruhe Institute of Technology (KIT). This first proof of principal introduced the

concept of "Horizontal Partitioning" [1]. In contrast to this older solution the concept and implementations presented here do not need any modification of the used batch system. By decoupling the batch client from the worker node, running the batch client on the hardware node but the worker node as VM (figure 2), the batch system can consider the VM as part of the users job. The VM management is handled by the prologue and epilogue hooks, which are part of almost all standard batch systems. Thus, the whole integration of virtual worker nodes into the batch systems is a matter of its suitable configuration and not a big software development issue.

2. Integration of Virtual Worker Nodes into Standard-Batch-Systems

Virtualized server nodes are increasingly used to consolidate the server infrastructure within high-availability clusters. Despite this, the task of using virtualization in a HTC cluster is quite new. Providing worker nodes with the operating system directly running on the hardware yields the best outcome in computing performance. This may be the usual case, but as discussed in the introduction there are situations where a user group can benefit from participating in a shared computing infrastructure so that the overhead of virtualization is negligible, see [1]. Here the dynamic partitioning of a cluster using virtualization enables the user groups to participate even if they have very special requirements for the computing environment.

The choice of operating system environment within high-level workflows such as Grids, is also enabled via virtualization, which will allow Grid infrastructures to support user communities without imposing a unified operating system. This will provide immediate benefit in facilitating staged migration from deprecated operating system releases and makes it easier to coordinate between both, experiments and sites. The potential exists to allow specific user groups to install directly a machine image within the virtual machine. User created operating systems are though outside the scope of this paper.

2.1. Choice of Virtualization Technique

As already discussed above, there are requirements for a virtualization technique in a dynamically partitioned HTC cluster:

- Support of commodity operating systems such as Linux or Windows.
- Acceptable overhead in performance.
- Easy installation and maintenance of virtual machines.
- Isolation of virtual machines from the host system.
- Controlling of virtual machines using scripts.
- Acceptable additional costs per existing physical host.

Xen[2] and KVM[3] are our candidates of choice for such a partitioning. While Xen was the standard for years now in common Linux Distributions, KVM with its minimal modifications to the Linux kernel is now becoming incorporated into most Linux Distributions.

Xen and KVM have a different approach to virtualization (see figure 3). Xen uses a hypervisor which is responsible for the scheduling and memory management as well as the hardware access of the virtual machines. In contrast to that, KVM uses the functions of the Linux Kernel. As a consequence the VMs appear now as a standard Linux process. Additionally, KVM depends upon the on hardware virtualization features of the CPU.

2.2. Using libvirt to Steer Xen and KVM Virtual Machines

Virtualization on Linux systems has been maturing in recent years and many different virtualization solutions are available in the Open Source community, each with its own

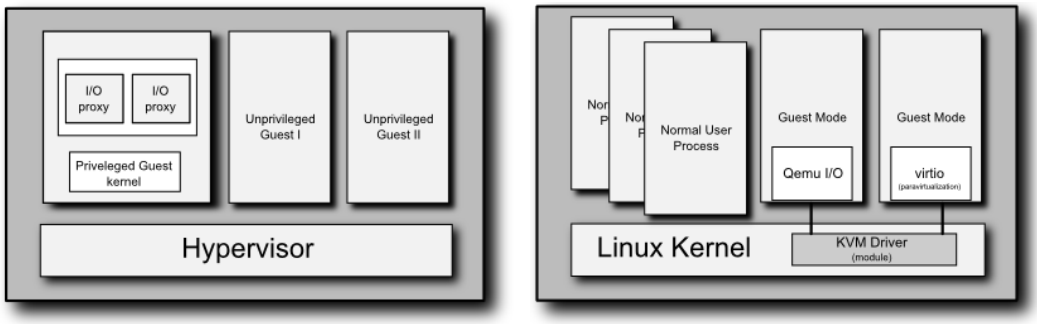

Figure 3. Common virtualization model using a hypervisor on the left hand side and Linux as a hypervisor on the right. The Linux Kernel schedules and manages the memory of the VMs. The hardware access is provided by the KVM kernel module and supports emulation and paravirtualization. KVM relies on hardware virtualization support of the CPU.

configuration and management tools. To enable compatibility between environments with different virtualization systems, libvirt [4] provides a common API for management and configuration of virtual machines. Both, Xen and KVM are supported as well as other virtualization solutions less focused on performance. The initial DESY implementation was bound to Xen management tools, but both projects decided to base the latest implementations upon libvirt. This enables to decouple the dynamic virtualization infrastructure from the virtualization layer and thus to switch easily between Xen and KVM and also to benefit from the functions and interfaces libvirt provides. In our cases we use the libvirt python bindings for the management scripts of the VMs. By using libvirt each VM is defined in a uniform XML configuration file, no matter which virtualization infrastructure is in use except the Hypervisor field which has to be adopted.

2.3. Preparation of the Virtualized Worker Nodes

A user can either request a certain software environment including already installed and configured computing software or prepare their virtual machine template on their own.

The virtual worker nodes will be started on the hardware nodes automatically. A very important setup step is to configure the network of the virtual machine. This is done via DHCP. Other configuration issues like the connection to the network or a cluster file system, can be completed within the virtual machine template before the initiation.

2.4. Encapsulation of Computing Jobs into Virtual Cluster Partitions using Prologue and Epilogue

A common functionality of batch or workload management systems are “prologue” and “epilogue” scripts which may be run before or after a job execution on the specific worker node. The standard usage of those scripts is the preparation or clean-up of the nodes. We decided to use this technique to encapsulate the original job, which has to be run within a virtualized worker node, in a so-called “wrap-job”.

Two different implementations of the integration of virtual worker nodes in standard batch systems are presented in the following. One with MAUI/Torque [5] as scheduler and batch system and one using SGE [6].

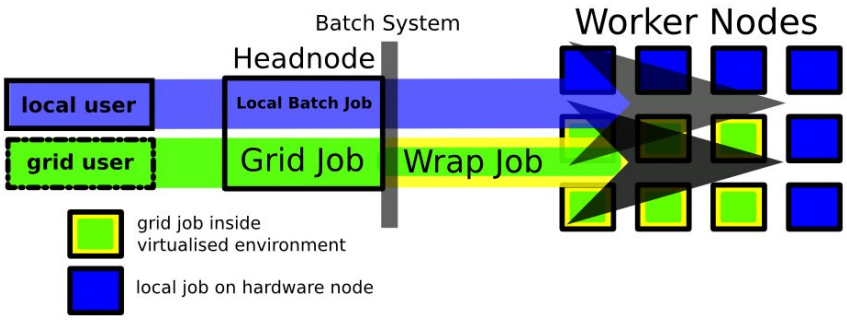

Figure 4. Workflow of the two different job types. Local jobs are executed on the hardware hosts, Grid jobs are wrapped and passed to a VM.

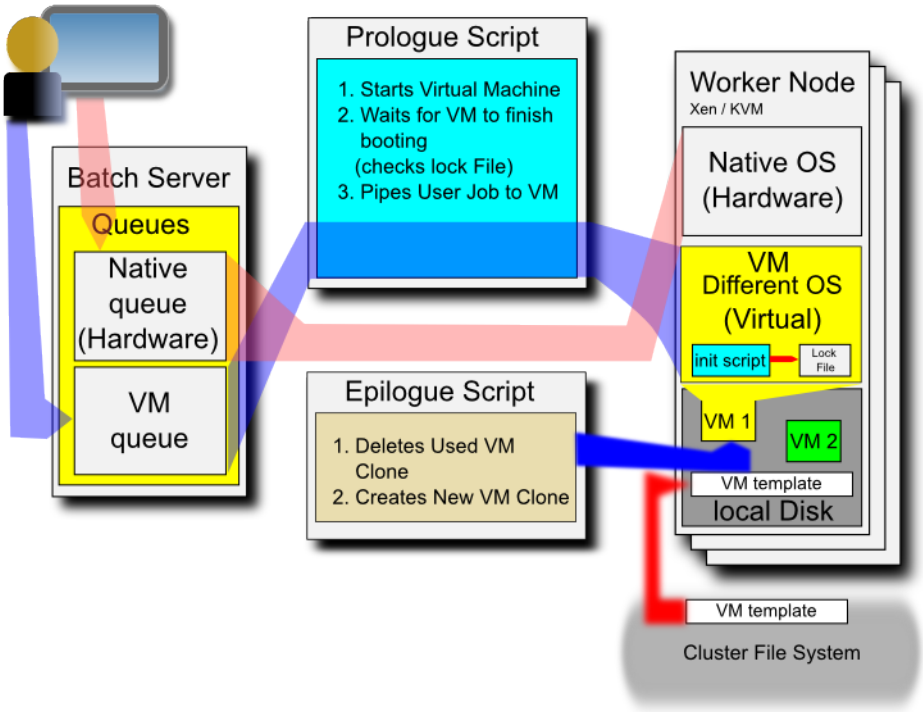

Figure 5. The wrap-job. It starts and stops the virtual machine on the designated worker node and passes the original job to it.

3. Implementation details of the two groups

The common features and working principles were presented up to now. This section will now specify the details of the two implementations.

3.1. MAUI and Torque at the Karlsruhe Institute of Technology

In the MAUI/Torque solution at the Karlsruhe Institute of Technology there are two types of available partitions on the cluster. The worker nodes of one partition are virtualized in contrast to the nodes in the basic partition that consists of the hardware nodes. Obviously, the wrap-jobs are only needed for virtualized partitions. In figure 4, the workflow of the two different job types is sketched.

Jobs submitted to the basic partition are executed as usual. In contrast to that, jobs that require

a different partition are encapsulated into prologue and epilogue scripts of the batch system. The prologue script starts the virtual machine on the designated hardware worker node as part of the job. As soon as the virtual machine is booted the original job is passed to the virtual worker node using a SSH pipe.
After the job completion the wrap-job will take care of the disposal of the virtual machine. If the job is interrupted in its execution by a system signal from the batch system, the termination of the virtual machine will be carried out via the batch system's epilogue script. The virtual machine will be instantly destroyed, as a clear shut-down is not necessary.
The boot time of the virtualized worker nodes depends on the OS started within the VM. In most cases it takes less than 30 seconds. This is negligible compared to the mean job execution time in our case. The Torque batch server provides access to different job specifications via arguments, which it passes to the prologue script (see Torque Documentation [5]). These specifications, for example the job id, are then used to start the virtual machine and pipe the job via SSH to the VM as sketched in figure 5.
The VMs are handled by a python script, which uses the libvirt python bindings to start a predefined VM. It is invoked by the prologue script previous to the job execution. It checks for a valid VM image, renames it to add the job id in the image name and starts the VM. As the boot-up process of the VM has to be finished before it can execute the users job, the python script ensures that all needed services on the VM are running like network mounts or logging. To provide this information, a very simple init script is added to the VM template which is executed at the very end of the boot phase. It creates a simple lock-file in the temporary directory. The python script waits for this file to appear on the VM and notifies the prologue script that the VM is ready to accept the job via its exit status.
As mentioned before, the epilogue script is automatically called by Torque after job completion to destroy and delete the used VM. It also generates a new VM image from a template on the local disk. The next job then renames this to add the needed job id in the file name and the process starts from the beginning.
The VMs are managed centralized. The main templates of the different VM types are placed in the cluster file system where one can manage updates and configuration. After a change these templates are replicated on each hardware host.

3.1.1. Testbed The following list shows the hardware used in the test setup:

- Number of test worker nodes: 4
- CPUs per node: 2xIntel Quadcore Xeons
- Disk space per node: 750 Giga Byte

The software and operating system configuration is detailed in the following list:

- Operating System on the worker nodes: SuSE Linux Enterprise Server 10sp2
- Operating System in the virtualized Grid partition: Scientific Linux 5.1 CERN Edition [7]
- Cluster file system: Lustre 350 Tera Byte
- Batch system: MAUI 3.2.6p19 , Torque 2.3.1
- libvirt 0.5.1

3.1.2. Grid Services at KIT The main reason for the partitioning of the shared cluster at the University of Karlsruhe was the effort to participate in the Worldwide LHC Computing Grid (WLCG) [8] as a Grid Tier 3 centre. Therefore it is necessary to run Grid middleware services. In our case, we need additional four servers to support the gLite [9] middleware infrastructure:

- Computing Element (CE): Generic interface to the local computing resources.

- Storage Element (SE): Provides uniform access to local data storage resources. Supports different data access protocols.
- Monitoring Box (MONBOX): Responsible for monitoring, accounting and bookkeeping.
- Site-BDII[1]: Collects information about all resources present at a site and distributes this to a top-level BDII which is queried by the Grid workload management system.

These services are also virtualized and put on a separate server infrastructure.

3.2. Implementation of SGE at DESY

The solution at DESY is conceptually equivalent to the previous one: The batch system SGE [6] also offers prologue and epilogue hooks. The SGE starter method modifies the starting of the job itself. In the DESY implementation, this method is used to execute the payload remotely via SSH on the virtual machine running on the same physical machine. The handling of the appropriate virtual machine is achieved during the prologue and the epilogue phase of the job. In the current implementation, a virtual machine is used for one single job, and is reinstalled by the prologue hook.
The management of the virtual machines is provided through VmImageManager [10]. This application was originally developed for deployment testing, and allows the reinstall of a running VM within 30-40 seconds. By default VmImageManger make use of rsync which synchronises the VM to a reference image, transfering only a minimum of data. VmImageManager can also archive parts of the virtual machine for the extraction of log files. In order to always have a mapping between a virtual machine and the SGE job, the job ID is passed to the prologue and epilogue method. VmImageManager offers a method to wait until the booting process of the virtual machine is finished. Then, the SGE starter method is used to copy the job through an SSH tunnel to the virtual machine and to execute it there. When the job is finished its output data are redirected to the WMS (see figure 2) and the virtual machine is stopped by calling a VmImageManager routine within the epilogue script.

3.2.1. Testbed

- One Computing Element (CE), with lcg-CE 3.1.18 installed on Scientific Linux (SL) 4.4
- One SGE batch server, with SGE 6.1u3 installed on SL 4.4
- Two physical hosts, each with dual-Xeon/3.0 CPU and 4 GB RAM. SL 5.1 is installed, running 2.6.18-53.1.21.el5xen kernel. No gLite middleware components installed, only the SGE processes execd and shepherd
- The Virtual Machines are based on SL 4.7, running an 2.6.9-68.7.ELxenU kernel, with glite-WN version 3.1.21-0 installed. No SGE binaries are installed.

3.3. Logging and Security

The job runs with user rights. To manage the VMs root access is needed. Therefore the users are sudoers for the start and stopping scripts of the VMs. The authentication on the virtual worker node is done via a suitable hosts.equiv to assure secure SSH login from the physical host to the VMs.
As each VM image is reset using different approaches after a job execution, this provides security to the job as artefacts from previous jobs cannot be used to compromise future jobs. The KIT and DESY teams both use different techniques for regenerating fresh virtual machine images but the outcome is equivalent. The logs of these VMs have to be stored to enable forensics for

[1] Berkeley Database Information Index

security maintenance and debugging.
The KIT team solves this problem using a log server to store the VM logs while the DESY team also provide a mechanism for storing extracted directories to archive the VM's log files while completing each reboot. The utilization of the central log server has to be configured within the virtual machine templates, but has the advantage that there is a single place to query all log files.

4. Related Work

Today, there are a variety of different techniques and implementations for virtualizing worker nodes. Each have their particular abilities and use cases. Primarily, a dynamic virtualization concept is meant to be used to allow a fully automated and thus dynamic partitioning of the hardware infrastructure of a cluster. This is in contrast to the way it is done in current virtualization management systems like the Amazon Elastic Compute Cloud (EC2) [11] or Eucalyptus [12] where a direct interaction of the user or the administrator is required. The prototype implementation presented in this paper, however, intrinsically includes the deployment of VMs in a cluster and even obeys fairshare policies without outside interference by a user or administrator. Other solutions using a simliar technique always required a batch system with the ability to manage VMs as batch system clients e.g. MAGRATHEA [13] or XGEv2 [14]. In our case the VMs are not seen by the batch server as a client but as a computing job. Therefore, the batch system does not have to be aware of virtualization, it just requires the prologue and epilogue scripting ability, which is the case in most of the common batch systems.

5. Conclusion, outlook and future work

When different groups with divergent operating system requirements want to share the same computing resources in a cluster, it might be partitioned. In order to intercept peak load demands in the different partitions, this sectioning of the cluster should be dynamic. Both implementations have successfully integrated with the WLCG Grid and allow the possibility to offer multiple operating systems to the users. It might be also possible to give the user the possibility to provide a customized VM image.

When partitioning a cluster, one should chose a scheme with the following features:

- Fast and simple response to change occupancy patterns.
- Offering an optimal configured operating system for every job class or the operation system of choice for each user.
- Leave the choice of the host operating system to the system administrator.
- Only minimal changes to the host operating system and the batch queue configuration.
- Performance impact due to virtualization is acceptable compared to the benefits.

The dynamic partitioning concept using virtualization presented in this article matches all these requirements.

We presented two implementation of dynamically partitioned clusters using KVM or Xen through the libvirt API on two test systems at KIT and DESY. Both implement the same concept of separating the batch client on the host system from the worker node, represented by a VM. In this case only minor changes to the configuration of standard batch systems with prologue and epilogue hook are necessary. The two batch systems are shown to provide the general functionality of this concept.

The next step will be to test both implementations in terms of scalability and stability in long term tests on large scale clusters. At the Karlsruhe Institute of Technology, we will test

our implementation on a machine with up to 1600 cores. After a successful testing phase we will put this virtualized cluster partition in our production environment of our WLCG [8] Tier 3 centre.

The ExecutionEnvironment object within the Glue Schema 2.0 [15] contains a description of the hardware and operating system for a homogeneous set of Worker Nodes. This object has an attribute called VirtualMachine which is set to TRUE if the job is intended to run on a virtual machines, thus allowing the Job Description Language (JDL) [16] to specify that it needs to run on a Virtual Machines. It is then conceivable that ExecutionEnviroments can be set to VO based images, or even user specific ones.

The latest releases of Blah and Cream [17] now provide the JDL information to the batch queue, enabling the assignment of memory and CPU cores to the virtual machine dependent upon the user requirements. This opens the use of gLite Grid deployments with virtualized Worker Nodes to communities that may require less homogeneous computing.

In our model, a potentially user or group defined image may be downloaded during the prologue and started with the job payload. This is technically envisageable, several security questions would however need to be addressed. The solution presented here seems to allow a cloud like infrastructure.

6. Acknowledgements

We thank the staff of DESY IT department, for the opportunity to pursue this research. We thank the staff of Steinbuch Computing Centre at the KIT that was responsible for the general setup and maintenance of the IC1 cluster.

We are greatly indebted to Andreas Haas from SUN/SGE for providing us with many ideas and insights on SGE.

We wish to acknowledge the financial support of the Bundesministerium für Bildung und Forschung BMBF and the Helmholtz Alliance "Physics at the Terascale".

References

[1] Volker Büge, Yves Kemp, Marcel Kunze, Oliver Oberst, and Günter Quast. Virtualizing a batch queuing system at a university grid center. In *LECTURE NOTES IN COMPUTER SCIENCE*, volume 2006, ISSU 4331, pages 397–406. Springer, 2006. 3

[2] XEN Virtualisation
`http://xen.org/`. 3

[3] KVM Virtualisation
`http://www.linux-kvm.org/`. 3

[4] Libvirt Virtualization API
`http://libvirt.org` 4

[5] The MAUI Scheduler
`http://www.clusterresources.com/` 4, 6

[6] SGE Policy-based Batch Queue
`http://www.sun.com/software/sge/` 4, 7

[7] ScientificLinux Homepage
`https://www.scientificlinux.org/` 6

[8] Worldwide LHC Computing Grid Homepage
`http://lcg.web.cern.ch/LCG/` 6, 9

[9] GLite Grid Middleware
`http://glite.web.cern.ch/glite/` 6

[10] VmImageManger - a Command Line Virtual Machine Image Management Script
`http://vmimagemanager.wiki.sourceforge.net/` 7

[11] Amazon Elastic Compute Cloud
`http://aws.amazon.com/ec2/`. 8

[12] Eucalyptus Cloud Infratructure
`http://www.eucalyptus.com/`. 8
[13] Miroslav Ruda, Jiri Denemark, and Ludek Matyska. Scheduling virtual grids: the magrathea system. In *USA,Second International Workshop on Virtualization Technology in Distributed Computing*, pages 1–7. ACM digital library, 2007. 8
[14] M. Smith, M. Schmidt, N. Fallenbeck, T. Doernemann, C. Schridde, and B. Freisleben. Secure on-demand grid-computing. *Journal of Future Generation Computer Systems*, pages 315–325, 2008. 8
[15] Common Information Model of Grid Entities
`http://www.ogf.org/documents/GFD.147.pdf` 9
[16] Job Description Language
`https://edms.cern.ch/document/592336` 9
[17] The Cream Computing Element and Blah a Component in the Glite Grid Middleware Stack
`http://grid.pd.infn.it/cream/` 9