Virtualized Batch Worker Nodes: Conception and Integration in HPC Environments

A Scheurer\textsuperscript{1}, O Oberst\textsuperscript{2}, V Büge\textsuperscript{1}, G Quast\textsuperscript{1} and M Kunze\textsuperscript{2}

\textsuperscript{1} Karlsruhe Institute of Technology (KIT), Institut für Experimentelle Kernphysik (EKP), Campus Süd, Postfach 69 80, 76128 Karlsruhe, Germany
\textsuperscript{2} Karlsruhe Institute of Technology (KIT), Steinbuch Centre for Computing (SCC), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany

E-mail: armin.scheurer@cern.ch

Abstract. There are various use cases where one has to separate user environments from the provided machine hardware, operating system or software setup on a cluster. To achieve this, a lightweight implementation of dynamically virtualized worker nodes in common batch systems is presented. Additionally, a summary of experiences and performance measurements obtained by running such a virtualized cluster, shared between nine different departments of the Karlsruhe Institute of Technology, is presented.

1. Introduction
During the past years, virtualization techniques evolved and today are a common and widespread mechanism to decouple software services and operating system (OS) from the underlying computer hardware. However, in the area of High Performance Computing (HPC), the virtualization of worker nodes still remains an unusual task due to the expected performance loss introduced by the additional virtualization layer between bare hardware and OS. This is in contrast to the currently rising popularity of Cloud infrastructures where the bare performance is short of the gained flexibility which such virtualized environments provide. But in some cases it is not possible to circumvent a virtualized setup even within a dedicated HPC cluster installation. Current High Energy Physics (HEP) experiments require specific operating systems, validated for their complex experiment-specific analysis software. Thus, the virtualization of worker nodes remains the only possibility for HEP communities to participate in and to benefit from shared HPC cluster environments.

This document describes ViBatch, a lightweight concept for dynamically integrating virtualized worker nodes in standard batch system infrastructures. It was developed and tested by using the open source scheduler and batch resource manager Maui/TORQUE \cite{maui}, \cite{torque} and the Kernel Based Virtual Machine hypervisor KVM \cite{kvm}. Since ViBatch only relies on the pro- and epilogue scripting functionality of the batch system and additionally, the hypervisor is addressed via the common virtualization API "libvirt" \cite{libvirt}, the whole concept can be easily ported to virtually any modern cluster infrastructure.

The next sections will give an overview of typical HPC cluster models and will present specific setup requirements of current large scale HEP experiments and collaborations. A developed benchmarking suite, optimized for typical HEP workflows, allows to compare the
performance of well defined test jobs, executed either within an operating system running on
the native computing hardware or inside a virtualized setup. The results of these performance
tests are discussed and potential drawbacks for HEP analysis workflows are depicted. Finally,
the integration of the virtual cluster concept in the production setup of a shared HPC cluster
environment at Karlsruhe Institute of Technology is evaluated and described.

2. HPC Cluster Models
Computing centers of universities and other large scale scientific institutions provide computing
infrastructures for various different and independent user groups. Each of these user groups
often have their very own and distinct requirements on hardware setup, used operating system
and software environment provided on the clusters. The following cluster models represent
possibilities to satisfy such individual demands:

2.1. Isolated Computing Cluster:
A still common but undesirable approach is to set up individual and separated clusters for
each working group. Such a setup allows virtually any possible operating system configuration,
however the gain in flexibility is directly negated by the huge arising administrative overhead.
Additionally, isolated clusters can not cover peak loads required during times of high demand.

2.2. Shared Computing Cluster:
In order to avoid this administrative overhead, a shared computer cluster is preferred. It requires
to find a compromise in terms of hard- and software setup between the participating groups but
in return allows to use the fair-share capabilities of local batch systems to cover peak loads in
case of high demands of an individual user or group. But in some cases it is not trivial to find
such a setup compromise as explained in more detail in section 3 due to specific contradicting
requirements of individual groups.

2.3. Dynamically Partitioned Cluster:
However, to still allow such user groups to participate in shared cluster environments, the concept
of a dynamically partitioned cluster has been developed and is presented in the scope of this
document. It allows virtually any operating system and software setup on a commonly used
cluster, but still benefits from load-balancing features of the used batch systems and as well
from a reduced administrative effort. Additionally, the virtualization layer remains hidden from
the user which allows transparent access to all provided resources. A more detailed description
of the concept is given in section 5.

3. Requirements of Current HEP Experiments
Typical HEP experiments like the Compact Muon Solenoid (CMS) [6] at the LHC [5] record huge
amounts of data for further analyses, which are performed by thousands of physicists distributed
all around the globe. The data volume accumulated during one year of LHC operation reaches
almost 2 PB in case of CMS and adds up to more than 10 PB in total, taking all four main LHC
experiments into account.

The distribution of such large datasets and the user access is done via Grid means. All
LHC experiments are connected hierarchically in a layered, four-tier structure which forms the
Worldwide LHC Computing Grid (WLCG) [7], mainly based on the gLite and OSG middleware
[8], [9]. gLite and as well the complex experiment specific analysis software require a dedicated,
verified and certified operating system to run on. The chosen OS is Scientific Linux CERN
Edition (SLC) [10], a Linux derivate based on RedHat. The main challenges of the CMS and
LHC computing model are: Grid middleware and experiment specific analysis software require a
dedicated operating system; high demands in terms of computing power and especially storage
resources; reliable access to the data via Grid means, ideally 24 hours, 7 days a week; additional
local computing resources for final end-user analyses; heterogeneous cluster configurations due
to the large collaborations.

4. Performance Testing with Typical HEP Workflows
In order to evaluate and quantify the actual performance loss due to the virtualization layer,
various synthetic benchmarking utilities are available.

Although they are perfectly suited to obtain performance results within their dedicated field
of operation, it remains hard to measure correlations between e.g. network and harddisk I/O
arising during standard HEP user analyses. For that purpose a dedicated benchmarking suite,
VMBench, was developed. VMBench uses standardized and well defined CMS analysis workflows
and evaluates the performance loss during each step of a typical HEP analysis chain. It provides
a modular setup, where so called "Watchers" are configured to measure e.g. used CPU time or
memory consumption for each individual step or the whole workflow chain in total.

All tests have been performed by running either locally within the host’s operating system or
inside virtual machines (VM) on the same hardware components. More information about the
underlying hardware setup is given in section 5.3. The results of the VMBench performance tests
are visualized in figure 1. Benchmarking results by running computing jobs natively within the
host’s operating system or a virtual machine are given in red and blue, respectively. In order to
perfectly match the ViBatch concept setup described in section 5, all virtual VMBench instances
were run in their own separated VM. The left plot gives the average relative running time of
a single VMBench instance when running 1, 2, 4 or 8 benchmark instances on a single 8-core
CPU host system in parallel. The right plot gives more detailed information depending on
the different workflow steps in case of running 8 benchmark instances in parallel. The most
time-consuming part of the workflow is the complex and CPU intensive simulation step which
for sure bears the largest absolute difference in running time for virtually and natively running
benchmarking jobs.

An interesting feature can be seen when comparing the last two columns labeled "SUM" and
"CHAIN" in the right plot. They represent the consumed time with and without intermediate
storage of the results of each individual workflow step on the clusters storage systems. As
expected, the absolute running time is increased when storing intermediate results on disk,
however the amount of additionally consumed time is larger in the virtualized case. This
indicates on a larger performance loss for disk I/O dominated workflows within a virtual setup.

However, the overall performance loss due to the virtualization layer stays below 10% and is
almost not affected by the number of running virtual machines on a single host in parallel. Due
to these results, virtualized worker nodes are a very good possibility for HEP experiments to
benefit from shared HPC cluster environments.

5. ViBatch
Within the last sections, the necessity for and the feasibility of using virtual worker nodes in
the scope of current HEP experiments inside HPC cluster environments has been shown. Due
to that, a lightweight concept of dynamically integrating virtualized worker nodes in standard
batch systems has been developed. The ViBatch concept allows real-time configuration of a
cluster setup according to current needs by dynamically deploying virtual machines on host
worker nodes. A first prototype implementation can be found in [11]. The concept bears the
following main features which are described in more detail below:

- Easily portable to virtually any modern batch system which provides pro- and epilogue
  scripting functionality.
Figure 1. Visualization of VM Bench results executed within the native host OS (red) and in virtual machines (blue) running on the same host. The average relative running time of one VM Bench instance has been evaluated for 1, 2, 4 and 8 running VM Bench instances in parallel on the same host system (left). Breakdown of the 8 VM case for the different HEP workflow steps (right).

Figure 2. Schematic overview of the ViBatch concept: portable to any batch system with pro- and epilogue scripting functionality, independent from the underlying hypervisor, lightweight setup, transparent to the user, allows a mixed batch system setup with native and virtual worker nodes.

- Batch system needs nothing to know about virtual machines, they are just part of a job.
- Independent from the available hypervisor by using the unified virtualization API: libvirt.
- Lightweight setup, only minor batch system configurations required.
- Transparent to the user.
- Allows a mixed batch system setup with native and virtual worker nodes in parallel.

5.1. The Concept
The ViBatch concept requires pro- and epilogue scripting functionality of the used batch system and a hypervisor supported by the libvirt virtualization API described briefly in the next subsection. A schematic overview can be found in figure 2. A detailed description of the conceptual workflow is given below:

(1) A user submits a job to a batch system. He decides if the job should run on a virtual worker node or the native host OS by submitting to an appropriate queue which needs to be set up...
on the batch server. Due to that it is easily possible to mix up virtual and native worker
nodes on the same cluster.

(2) If a virtual queue is selected, the batch system executes the prologue script at the beginning
of each job.

(3) The prologue script prepares the virtual machine image by cloning the VM from a provided
template on the local worker node hard disk.

(4) The template is modified to accept the actual user job later on. Currently, a user specific
public ssh key is copied to the authorized keys file on the VM. This guarantees that a user
job can’t access other users worker nodes running at the same time.

(5) Afterwards, the virtual machine is started via the libvirt API and a proper MAC address
for the virtual network interface is handed over to allow individual network setup via DHCP.

(6) At the end of the booting process, the VM creates a lockfile via an init script on the local
or cluster file system.

(7) The prologue script checks for this lockfile to guarantee a completely booted VM.

(8) The actual user job is piped via ssh to the VM.

(9) The user jobs is executed inside the VM.

(10) After the job has finished and the job output was returned to the user, the epilogue script
is executed.

(11) The epilogue script does some cleanup like shut down of the VM and network bridges,
afterwards the VM image is destroyed.

A more detailed documentation of ViBatch and as well technical information about the
implementation like required configuration changes of the batch system and necessary script
files can be found in [12].

5.2. Common Virtualization API: libvirt

Today, various virtualization techniques are available. In order to be independent from the
available hypervisor on the cluster, a common virtualization API was chosen to access the
underlying virtualizer from within ViBatch. This common API is called libvirt and supports
most modern virtualization solutions like KVM, Xen, OpenVZ, VMware and VirtualBox. The
management of the virtual machines is either possible via a command line interface ("virsh"
or a graphical user interface ("virt-manager"). libvirt provides bindings to various scripting
and programming languages like C, C++, Python, Perl, Ruby and Java. Another advantage
of libvirt is its encrypted remote management capability which currently supports TLS, x509
certificates, Kerberos and SASL authentication.

5.3. Implementation and Operation

The ViBatch concept has been integrated into the production system of an HPC cluster located
at Karlsruhe Institute of Technology. The cluster is shared among nine different institutes and
provides the following key specifications: 1600 CPU cores, 200 x 8-core Intel Xeon X5355 (VT-x,
64bit) CPUs with 2 GB of memory per core. The host OS is SUSE Linux Enterprise Server 10.2
(Kernel 2.6.16) with the KVM hypervisor (version 88).

The current implementation has two virtual machine queues configured, one with a guest OS
based on Scientific Linux CERN 5, which is patched to Kernel 2.6.31, and the other one running
a standard Debian [14] distribution version 5.0/Lenny. During our tests, more than 25 000 user
jobs have been submitted to the Maui/TORQUE batch system. The jobs tested all aspects
necessary for a proper running of HEP applications within the cluster like CPU intensive Monte
Carlo simulation or I/O intensive reconstruction jobs.
To date, no para-virtualized driver was available for direct access of the Lustre[13] cluster file system from within a virtual machine. Thus, a scalable solution for storage access has been implemented where each host system provides an NFS export of the natively connected InfiniBand file system to locally running VMs via virtual network interfaces. Additional stability and robustness tests have been performed, simulating e.g. worker node and batch system crashes, which were handled without problems by ViBatch.

6. Summary and Future Plans

Current HPC clusters are often shared between different institutions and user groups. In order to participate in such a shared environment, a compromise in terms of hard- and software setup has to be found. In some cases, however, it is not trivial to find such a compromise. High Energy Physics experiments, for example, require dedicated operating systems for their Grid and complex analysis software. Thus a participation of HEP groups in a shared cluster environment is only possible if this cluster runs the required OS natively or virtualized worker nodes are available.

The ViBatch concept allows to dynamically integrate virtualized worker nodes into standard batch systems in a lightweight and transparent way. The concept is independent from the available batch system as long as it provides proper pro- and epilogue scripting functionality. Due to the usage of libvirt, a unified virtualization API, the concept can be easily ported to different hypervisors supporting it. ViBatch has been deployed on a 1600 core cluster located at Karlsruhe Institute of Technology, which is shared among nine different institutions. Large scale tests with more than 25000 user jobs have been performed and its stability and robustness against worker node crashes has been verified. The whole setup is running in parallel to the production environment with mixed native and virtual batch queues.

In its current implementation, ViBatch requires the storage of the virtual machine images on a local hard disk of each worker node. An automatic image deployment is currently under investigation for future releases of ViBatch. Additionally, an improved authorization mechanism for access to the virtual instances by the user via disposable ssh host keys is under development.

Acknowledgments

We want to thank all involved EKP and SCC members of KIT Campus North and South for their support and help during the development and implementation of ViBatch. In addition we thank the "Bundesministerium für Bildung und Forschung" (BMBF) as well as the Helmholtz Alliance "Physics at the Terascale" for their financial support.

References

[1] Maui Cluster Scheduler, http://www.clusterresources.com/products/maui
[2] TORQUE Res. Manager, http://www.clusterresources.com/products/torque
[3] Kernel Based Virtual Machine, http://www.linux-kvm.org
[4] libvirt: The Virtualization API, http://libvirt.org
[5] The Large Hadron Collider, http://lhcb.web.cern.ch/lhcb
[6] The Compact Muon Solenoid, http://cms.web.cern.ch
[7] The Worldwide LHC Computing Grid, http://lcg.web.cern.ch/lcg
[8] gLite - Middleware for Grid Computing, http://glite.web.cern.ch/glite
[9] Open Science Grid, http://www.opensciencegrid.org
[10] Scientific Linux CERN, http://linux.web.cern.ch/lucens
[11] Prototype Concept: ViBatch, Volker Buege et al. 2010 J. Phys.: Conf. Ser. 219 052010, doi: 10.1088/1742-6596/219/5/052010
[12] ViBatch Wiki, https://ekptrac.physik.uni-karlsruhe.de/trac/BatchVirt
[13] Lustre File System, http://www.lustre.org
[14] Debian Linux, http://www.debian.org