Long-term preservation of analysis software environment

Dag Toppe Larsen, Jakob Blomer, Predrag Buncic, Ioannis Charalampidis, Artem Haratyunyan
CERN PH-SFT, Geneva, Switzerland
E-mail: Dag.Larsen@cern.ch

Abstract. Long-term preservation of scientific data represents a challenge to experiments, especially regarding the analysis software. Preserving data is not enough; the full software and hardware environment is needed. Virtual machines (VMs) make it possible to preserve hardware “in software”. A complete infrastructure package has been developed for easy deployment and management of VMs, based on CERN virtual machine (CernVM). Further, a HTTP-based file system, CernVM file system (CVMFS), is used for the distribution of the software. It is possible to process data with any given software version, and a matching, regenerated VM version. A point-and-click web user interface is being developed for setting up the complete processing chain, including VM and software versions, number and type of processing nodes, and the particular type of analysis and data. This paradigm also allows for distributed cloud-computing on private and public clouds, for both legacy and contemporary experiments.

1. Introduction
There exists a wide variety of computer platforms, differing both in terms of architecture, i.e. CPU type, and in terms of generation, i.e. evolution within the same architecture. In both cases, there are challenges with regards to transferring computer software originally developed for one platform to another. If the software is developed in a low-level language, i.e. assembly, the transfer to another architecture will usually require a complete rewrite. To remedy this, high-level languages, such as FORTRAN and C/C++, were developed to allow compilation on several platforms. This approach proved to introduce challenges by itself. Hardware vendors added extensions optimised for their own architecture. Operating systems provided incompatible system libraries. Compiler vendors used differing syntax. In some cases, the programming languages are not supported on all platforms. Even within the same architecture, the evolution of hardware, operating system, system libraries, and compilers, meant that source code compatibility from one generation to the next was not assured. In recent years, great progress has been made with respect to harmonising programming language syntax and system libraries across different platforms and vendors. However, especially with respect to the evolution within a given platform, it is difficult to achieve such harmonisations, since evolution is in fact needed as new technologies are developed. Consequently, while high-level languages did ease the task of transferring computer software from one platform to another, it did not entirely erase the border between different platforms, still requiring substantial manual work for the migration.

Computer software represents a huge investment. Even disregarding issues such as security threats to legacy, unmaintained platforms, one can not choose not to transfer the software to a
new platform (assuming the task the software performs is still needed). The software depends on compatible system libraries and compilers, which again depends on compatible operating system, which again depend on compatible hardware. At some point compatible hardware will no longer be available, and the whole dependency chain collapses.

This is of course not a unique challenge to the HEP community. It exists for all users of computing resources. What may constitute, if not a unique challenge, at least an unusual challenge, is the long life span of HEP experiments. For example, it took 20 years to plan and construct the LHC experiments, with another 20 years of expected running time. Also, many smaller experiments have long life-times, in the sense that an old experiment may evolve into a new one. There will be several decades of data-taking (including simulations from the planning stage) that may be of interest to re-process at a later date. In addition, the desire to keep a historical record may suggest preserving the data “forever”. While one can hope that software undergoing active development has a current version that is compatible with contemporary computing platforms, one can not assume the same for a 20-year old version of the same software. This is important for two reasons: (1) during those 20 years of operation, the experiment may undergo substantial upgrades and modifications; at some point the current software version may not be able to process legacy data, and one has to rely on legacy software versions for this task; (2) if the current version of the software is still capable of processing legacy data, it is very useful to compare the output from newer and older software versions as part of the validation of the results.

Traditionally, data preservation has often referred to the preservation of just the data byte stream itself, together with the source code and/or binaries used to process them. Practically speaking, this means transferring a set of files to newer storage technologies before the old storage system becomes obsolete. Although this task may have a non-negligible cost, it is practically feasible, especially on modern bulk storage systems. This idea of data preservation may be appropriate for data that can easily be interpreted without complex processing. However, data from HEP experiments rarely fall into this category. Detailed knowledge about the detector and the data taking conditions is necessary for the calibration and reconstruction. Without this, the data are useless. Often, much of the knowledge and know-how about this procedure is embedded in the software itself. Transferring the software to a new platform not only requires substantial effort, it also introduces a substantial risk that new bugs are introduced, either as a result of trivial coding mistakes, or because the code is not properly understood, thus wrongly modified to fit the new platform. Discovering such mistakes requires tedious validation of the outputs from the new and old platforms.

Thus, it is possible to argue that true data preservation also requires the preservation of the full operating environment. This can be achieved using virtual machines (VMs), which allow for legacy operating systems to run in a VM, with virtualised legacy hardware. Until recently, virtualisation was “exotic” technology only used in very specialised computers. However, in recent years, virtualisation has become increasingly popular and mainstream, consequently making this approach a more viable option for preservation of the analysis software environment. Thus, the operating environment may also be preserved as a file, containing the disk image of a complete operating system, given that the hypervisor-interface does not change.

2. CernVM-based data preservation

CERN virtual machine (CernVM)\(^1\) \([1, 2, 3]\) is a distribution of Linux, targeted specifically for VMs for scientific data processing. It has also initiated the development of a set of related technologies, which, although in principle independent, are collectively considered part of CernVM. Although CernVM was created as a way to perform ordinary production data

\(^1\) http://cernvm.cern.ch
processing for HEP experiments, it also exhibits excellent properties for the preservation of the software environment. The key point is that if one already has deployed a full-scale virtualised processing framework, it will also be possible to launch the exact same VMs in 20 years, running exactly the same software, processing exactly the same data, obtaining exactly the same results. If the CernVM based, virtualised processing is deployed appropriately, the data preservation can come as part of the “package” (of course, some know-how about the old versions of the software is required). The CernVM technologies relevant to the data preservation will be described in detail in the sections below.

3. CernVM virtual machine

The CernVM virtual machine is the corner-stone of the project. Although it was developed with the needs of the HEP community in mind, it is also suitable for general, virtualised processing. It is currently based on Scientific Linux (SL)\(^2\). The close relationship assures compatible results from CernVM and SL 5 processing. CernVM versions based on newer versions of SL will be made available as the general CERN processing is also moved to newer SL versions. However, legacy versions of CernVM can still be regenerated for legacy versions of the analysis software, i. e. for data preservation usage. Moreover, CernVM comes in both 32 and 64 bit versions.

To minimise the size of the file system images, only the essential software is included. This makes it possible to distribute CernVM as compressed images of about 300 MB. Additional operating system components are installable via the Conary\(^3\) package manager. Experiment software, including associated software, is distributed via CernVM file system (CVMFS) [4, 5, 6, 7].

CernVM comes in different editions, matching different use cases. The two main use cases are batch system and developers’ platform. The batch system images include a Condor\(^4\) batch system, and can be contextualised through the Amazon EC2\(^5\) interface. Amazon EC2 is the de-facto standard for contextualisation of VMs, not only on public clouds like Amazon EC2, but also on most private clouds based on frameworks like OpenNebula\(^6\), OpenStack\(^7\), Eucalyptus\(^8\), etc. The contextualisation involves setting up and configuring the VM for the tasks it will be performing, such as installing Conary packages, enabling CVMFS repositories and setting environment variables needed for the specific tasks. Two editions exist of the batch images, i. e. batch node and head node. The developers’ platform images are intended to be used as VMs on the developer’s laptop. If the development and the batch processing are performed in different environments, it is possible that a job will fail on the batch system even if it succeeded on the development system. Developing in the same (CernVM) environment as the batch system should significantly reduce this possibility. The developer images come in two editions, desktop and basic, with and without the X window system, respectively. Instead of Amazon EC2 contextualisation, they come with a local web interface which can be used to configure the VM. Further editions can be created for specialised use cases.

The trivial solution for preserving the operating environment, is to preserve the file of the disk images of different versions of an operating system. Indeed, this approach is also possible with CernVM. However, one can not know which hypervisors\(^9\) will be available decades from now. It may be they will pose requirements that are not compatible with the disk image files for current hypervisors. Hence, the disk images can not be used without prior modification.

\(^2\) http://scientificlinux.org
\(^3\) http://wiki.rpath.com/wiki/Conary
\(^4\) http://research.cs.wisc.edu/condor
\(^5\) http://aws.amazon.com/ec2
\(^6\) http://opennebula.org
\(^7\) http://openstack.org
\(^8\) http://eucalyptus.com
In the case of CernVM, a special tool, iBuilder [8], allows the usage of a “recipe” to create the disk images for the different CernVM editions, tailored to the specific hypervisors. Each version of CernVM has a unique top-level group version, called the **version string**. All operating system components and packages are strictly versioned and stored in a database as Conary packages. A version string refers to a strict group of such packages. Using this, iBuilder can exactly recreate a specific version of a CernVM edition for any supported hypervisor. This means it should be possible to create disk images for current and past versions of CernVM for future hypervisors that do not yet exist, but that will be supported by iBuilder at some point in the future. Of course, there is the caveat: how can one assure that this will not fall victim to the exact same process of obsolescence that the analysis software is experiencing? For iBuilder, one assumes that it will be an ongoing project, updated for new platforms and hypervisors as needed. For the legacy operating system, the only interaction it has with the outside world, apart from the hypervisor and the experiment software accessed via CVMFS, is the network access to the data to be analysed. In principle, there are no restrictions as to which protocol to use for the access; HTTP, CVMFS, **FTP**, **NFS**, **XRootD** and **EOS** are some of the possibilities. The access may either take place through a **FUSE** file system module (e. g. EOS), or through a stand-alone copy tool (e. g. **XRootD xrdcp** copy tool). Over time, protocols may evolve, or perhaps not be available at some point in the future. However, since FUSE file system modules and stand-alone copy tools typically only require a small and well-defined subset of the operating system features, it should be a reasonable task to support them on legacy CernVM versions.

4. CernVM file system

CVMFS is a distributed file system based on the HTTP protocol. The usage of HTTP also implies that it is globally distributed, even if there is no “local” support for CVMFS. Moreover, firewalls usually allow HTTP traffic (port 80) to pass through. For the software used by the CERN experiments, the “master” repository, called *stratum 0*, is a web server placed at CERN. Replication and load balancing is achieved through a global hierarchy of higher-order strata and HTTP proxy servers. Typically, every major site relying on CVMFS has a local proxy server to improve performance.

CVMFS is centred around repositories. A repository is a name space in the CVMFS directory structure, reserved for a specific entity, typically an experiment. Within this repository, the users are free to develop a directory tree suited to their needs. Also, there can be repositories providing shared software, e. g. **ROOT** is provided by the repository of the CERN **physics software** (PH-SFT) group. In the root directory of any computer running the CVMFS client, there is a directory */cvmfs*. Under this, all individual repositories are mounted. To avoid name space pollution, each repository is qualified with a “domain” indicating their origin. For example, all CERN-related repositories are post-fixed by **.cern.ch**. This gives full paths like */cvmfs/sft.cern.ch*, */cvmfs/na61.cern.ch* and */cvmfs/atlas.cern.ch*. It is the configuration of the client which decides which repositories are available.

To minimise network traffic and storage, all files on CVMFS are compressed on the server side. Further, a hash key is calculated for all files. The compressed files are stored on a common HTTP data storage, where they are referenced by their hash keys. Since identical files have the same hash key, this effectively prevents duplicates of the same file to be stored. The directory catalogue is stored in a **SQLite** database. This database is downloaded when the repository is mounted. In the case of large directory trees, it is possible to reduce the size of the catalogue by

---

9. [http://xrootd.slac.stanford.edu](http://xrootd.slac.stanford.edu)
10. [http://eos.cern.ch](http://eos.cern.ch)
11. [http://fuse.sourceforge.net](http://fuse.sourceforge.net)
12. [http://root.cern.ch](http://root.cern.ch)
13. [http://sqlite.org](http://sqlite.org)
using nested catalogues, typically one for each of the sub-trees containing a software release. For all files in the repository the catalogue contains the corresponding hash key. Whenever a file is opened for the first time, the file of the matching hash key is downloaded from the nearest proxy server, decompressed and stored semi-permanently to a disk cache. The cached file is usually not deleted, unless there is shortage of disk space. Hence, the file is typically only downloaded once, and it is also possible to run the software in off-line mode, i.e. without Internet connection.

Updates to the repository directory structure are performed on the central server. New files will be put on the HTTP file storage, and an updated directory structure database is created. Typically, files will not be deleted from CVMFS, only new versions of the software or calibration files are added to the repository. This means that legacy versions of the software will be preserved for usage in conjunction with the legacy CernVM images, i.e. enabling the preservation of the software environment for as long as the repository content is protected.

5. CernVM and private clouds

A cloud infrastructure is a prerequisite for virtualised computing facilities. Since CernVM supports all popular hypervisors, it can also run on a very wide variety of cloud infrastructures, both public (e.g. Amazon EC2 and RackSpace) and private (e.g. OpenNebula).

While the relative effort of setting up the infrastructure for a private cloud may be negligible for a large tier 0 or 1, it may be substantial for a small tier 2 or 3 with limited resources. To ease to deployment effort for tier 2 and 3, a reference cloud has been set up. It is based on SL 6, and uses KVM as hypervisor. Like most private cloud infrastructure, it supports the Amazon EC2 interface. This allows it both to behave as an EC2 cloud (accepting VMs to be launched on local hardware through the EC2 interface), as well as launching VMs on EC2 clouds. Detailed step-by-step instructions will allow sites to replicate the reference cloud with limited effort.

CernVM-based data preservation relies on VMs running legacy operating system versions. These systems are obviously not updated, and may have well-known security exploits. Thus, it is essential to run them in a protected environment, i.e. on a private network, separated from the general Internet with a firewall or network address translation (NAT). Only access to CVMFS repositories and experiment data should be allowed through. This may not be possible on all public clouds. In such cases, a private cloud is necessary to facilitate the protected environment.

Virtualisation also opens up new possibilities with regard to distributed computing. While HEP-experiments have traditionally relied on grid-frameworks for these tasks, virtualisation makes it possible to create transparent, distributed, virtual clusters of “identical” virtual worker nodes, spanning multiple physical computing sites. Hence, computing resource aggregation can be achieved without the involvement of grid software. This gives the sites the choice to either allow its resources to be part of a larger cloud through the EC2 interface (bypassing grid middle-ware), or launch VMs to process jobs directly themselves (using grid middle-ware). The limited resources available to tier 2 and 3 again might make it beneficial for them to only supply VM infrastructure to a larger tier 1, which will be responsible for setting up the proper grid middle-ware (which will submit jobs to the distributed, virtual cluster). The tier 1 can then transparently include the resources of the tier 2s and 3s in its local resource pool.

6. Workflow interface

There are many steps of processing from the raw data, as collected by the experiment, to the final physics plot, as published in a paper. One refers collectively to these steps as the workflow. Usually, the workflow is carried out by a mixture of manual and automatic processes.

14 http://rackspace.com
15 http://linux-kvm.org
16 http://cernvm.cern.ch/portal/privatecloud
Obviously, an experiment with large data sets typically requires a higher level of automation. The bookkeeping database contains the properties of all collected experiment data, and is a very important tool to the workflow management. It is essential to any large experiment. For an HEP experiment, relevant properties may include run number, time, beam energy, target type, quality ("good"/"bad"), reconstruction software version, operating system version, path to data, etc. It may contain both raw and reconstructed data. Also, an infrastructure for the processing is needed. Typically, this consists of a batch system, where the processing is distributed over multiple worker nodes. The initiation of the processing may be a manual task, where the user makes job description files to be submitted to the batch system by hand, or it may be fully automatic, where the relevant data is selected from some interface, and the job description files are created by the system.

There is a very tight interplay between these different components. Before processing data, the data must be selected from the bookkeeping database. The paths of the data are extracted, and inserted into the job description files. When the processing is completed, the bookkeeping database has to be updated to reflect this. Although it is possible to do these tasks by hand, it may be highly advantageous to automate them. Besides being time-consuming, it also introduces the risk of human error. For an automated system, the part responsible for this coordination is the workflow manager. The manager will allow the user to define a set of dependencies between the different steps of the processing. It also provides the interface where the user selects data, and optionally displays the processing status.

A “stricter” version of a workflow manager is called a provenance system. The provenance system keeps track of how all entities are created. In this context, an entity can be any file created from other files, such as a plot or reconstructed data. To take a far-reaching example, one can imagine a plot in a physics paper to be marked with a tag. If this tag is input to the provenance system, it will be able to recreate the plot (assuming the software and raw files still exist). A database will detail which files of reconstructed data, and which versions of the software and operating system were used to create the plot. Further, for each of the files of reconstructed data, the provenance system will also know which raw files and calibration files, and which versions of the software and operating system were used. Thus, a dependency chain can be constructed, and the plot automatically created. The step of actually creating the plot will require some custom code, containing cuts, etc., to be executed. This will have to be input manually to the provenance system, hence requires some “burden” on the side of the user. Except for this, all other steps to create a specific set of reconstructed files from given software and operating system versions and sets of raw files can be entered into the provenance system by itself.

A main new development of the CernVM ecosystem is a data flow management interface with integrated bookkeeping database and creation of job description files. CernVM already has a Condor batch system back-end. For an experiment basing the processing on CernVM, it is very beneficial to store the exact version string for the disk image used for the processing. As explained above, this string can be used to recreate the disk image at any point in the future, also for future hypervisors. Thus, it is also possible to recreate exactly the same processing environment on hypervisors that do not yet exist. There are several workflow interfaces available that either allow for management of VMs (e.g. Elasticfox\(^\text{17}\)), or that allow for management of experiment data using traditional batch systems (e.g. Ganga\(^\text{18}\)). However, there are few, if any, interfaces which combine these two aspects. Based on this, we propose a workflow interface that can: (a) keep track of the processing status of all data; (b) select data to be processed and which type of processing, also selecting the version of software and matching VM; and (c) the configuration of the processing nodes (e.g. number of nodes and RAM). Preferably, this will be

\(^{17}\) http://elasticfox.sourceforge.net

\(^{18}\) http://cern.ch/ganga
Figure 1. Processing data on a CernVM cloud using the workflow interface.
based on an existing system, and may or may not also be a provenance system.

Figure 1 shows a sequence diagram for how this is envisaged. The diagram is valid both in the context of “ordinary” processing and data preservation processing. The first step is similar to traditional bookkeeping systems, where the user selects the data relevant for the processing, based on certain criteria (e.g. energy and target). Next, the environment has to be set up. By default, the workflow interface will propose the current software version, and a version of CernVM matching the software version. The user may choose to accept this, or to select another software version, and possibly another CernVM version. For the configuration of the virtual cluster, the workflow interface will propose a set of default values. Again, the user may choose to accept the default, or modify the parameters. The user is now ready to submit the data for processing. After submission, the workflow interface will determine the relevant protocol and associated parameters to access the experiment data. The workflow interface has to check whether a suitable virtual cluster already exists. If not, a new cluster has to be created. This again will require a CernVM image matching the version string and data access protocol. If this does not exists, a new one has to be built by iBuilder. The needed packages will be requested from the CernVM package repository, and a new image is built. Subsequently, this is uploaded to a cloud using the EC2 interface. The workflow interface can now create the contextualisation files, and use these to launch the virtual cluster, also via the EC2 interface. After the virtual cluster has been created, jobs can be submitted to the cluster. Job description files are created for the batch system. As part of the job, the worker nodes of the virtual cluster will fetch the files needed for the processing via the appropriate data access protocol. The experiment software is accessed via CVMFS, and files are downloaded as needed. Typically, all files needed are downloaded during the first job, and will not have to be re-downloaded for subsequent jobs. If the job finishes successfully, the processed data will be uploaded to the data repository, again via the appropriate data access protocol. If the job is unsuccessful, it will be scheduled for reprocessing, and a new job description file is created for it. This will be repeated a number of times before finally giving up. When all jobs are finished, the user is notified. At this point, the workflow interface may either retain the virtual cluster for further jobs, or destroy it. The CernVM image may or may not be destroyed as well.

7. Status
Both CernVM and CVMFS are currently used by a number of LHC and other experiments in production. CVMFS is also used by a number of non-virtualised computing centres for software distribution. The reference cloud and the workflow interface represent new developments in the CernVM family of software components. The reference cloud is set up, currently in prototype status. Currently, the workflow interface is in an early stage of development. Production systems are expected by the end of the year.

References
[1] Buncic P et al. 2008 Proceedings of Advanced Computing and Analysis Techniques in Physics Research (ACAT) URL http://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=70
[2] Buncic P et al. 2010 J. Phys.: Conf. Ser. 219 042003 (Preprint doi:10.1088/1742-6596/219/4/042003)
[3] Buncic P et al. 2011 The European Physical Journal Plus 126 (Preprint doi:10.1140/epjp/i2011-11013-1)
[4] Blomer J et al. 2010 Proceedings of Computer Communications and Networks (ICCCN) (Preprint doi:10.1109/ICCCN.2010.5560054)
[5] Blomer J et al. 2011 J. Phys.: Conf. Ser. 331 042003 (Preprint doi:10.1088/1742-6596/331/4/042003)
[6] Blomer J et al. 2012 Proceedings of International workshop on Network-aware data management (NDM) 49–56 (Preprint doi:10.1145/2110217.2110225)
[7] Blomer J et al. 2012 Proceedings of International Conference on Computing in High Energy and Nuclear Physics (CHEP)
[8] Charalampidis I et al. 2012 Proceedings of International Conference on Computing in High Energy and Nuclear Physics (CHEP)