A Validation System for Data Preservation in HEP

Yves Kemp[1], **Marco Strutz**[2], **Hermann Hessling**[2]

[1]DESY, Notkestrasse 85, D-22607 Hamburg
[2]HTW Berlin, Wilhelminenhofstrasse 75A, D-12459 Berlin

E-mail: yves.kemp@desy.de

Abstract. Preserving data from past experiments and preserving the ability to perform analysis with old data is of growing importance in many domains of science, including High Energy Physics (HEP). A study group on this issue, DPHEP, has been established in this field to provide guidelines and a structure for international collaboration on data preservation projects in HEP.

This contribution presents a framework that allows experimentalists to validate their software against a previously defined set of tests in an automated way. The framework has been designed with a special focus for longevity, as it makes use of open protocols, has a modular design and is based on simple communication mechanisms. On the fabrics side, tests are carried out in a virtual environment using a cloud infrastructure. Within the framework, it is easy to run validation tests on different hardware platforms, or different major or minor versions of operating systems. Experts from IT or the experiments can automatically detect failures in the test procedure by the help of reporting tools. Hence, appropriate actions can be taken in a timely manner. The design and important implementation aspects of the framework are shown and first experiences from early–bird–users will be presented.

1. Introduction

Data preservation is a topic of increasing importance. This is true in general for any scientific data in the digital era, and it is especially true for High Energy Physics data. This has been recognized by ICFA, the International Committee for Future Accelerator, as they endorse since August 2009 the "Study Group for Data Preservation and Long Term Analysis in High Energy Physics", DPHEP[1]. As the DPHEP initiative is presented elsewhere in these proceedings, we simply summarize the main objectives of the initiative:

- Review and document the physics objectives of the data persistency in HEP.
- Exchange information concerning the analysis model: abstraction, software, documentation etc. and identify coherence points.
- Address the hardware and software persistency status.
- Review possible funding programs and other related international initiatives.
- Converge to a common set of specifications in a document that will constitute the basis for future collaborations.

In the context of the DPHEP initiative, many different aspects of data preservation and adjacent topics are discussed[2, 3]. In this contribution, we focus on the question how to preserve best the ability for performing analysis on preserved data. We start with some general considerations

from the perspective of the HERA experiments at DESY and, then, present a framework for preserving software developed at DESY.

2. Conserving analysis capability

When faced with the question "How long should we be able to analyze old data?", a universal answer cannot be given. There are of course scientific rules imposed by Science bodies or journals, but these are usually rather lose, do not specify the detail level of reproducibility, do not ask for new analyses to be done with old data, and sometimes do not extend to more than five years. In discussions among different experiments, we can find two categories:

- Experiments for which a superseding experiment is on the horizon
- Experiments for which no superseding experiment is on the horizon.

It is clear that in the first case, the interest in long term analysis is reduced: Often people want to be able to analyze data of their old experiment until new and potentially better data is available from the new experiment. Eventually, comparisons will be made between old and new data, but after some period of overlap, the old data and its analysis is felt obsolete.
In the second case, long term analysis is crucial: The collected dataset is unique, it should be preserved as well as the knowledge how to analyze it – potentially for eternity. This is clearly the case for the HERA data: No e^-p or e^+p collider is currently running and there are no serious dicussions for planning such a collider. Probably the most tangible plan would be LHeC, for which a "Conceptual Design Report" is being written.
It is, therefore, clear that the strategy for HERA concerning analysis preservation must differ from other experiments.
We propose an analogy with the real world: "Pizza preservation".

2.1. Discussion strategies: "Pizza preservation"

If the task is to preserve a pizza, the timeline plays a role. If we want to preserve it for a short time (in this case, some days up to some months), a freezer or deep freezer is a solution known to work. For example, the BaBar collaboration has set up an elaborated system for archival, more details can be found in several presentations[4].
If the task is to preserve a pizza for a longer time, conserving the pizza as a material entity is not a good solution. The timescale is simply too long to ensure integrity and constance of the result. Instead, it is advisable to preserve the recipe of the pizza. To ensure integrity and constance, it is important to bake the same pizza repeatedly, check for deviations - and potentially fix them. Along this line, we would propose a similar scheme for preserving the HERA software: Experiments would not preserve ready–made executables or OS images, but rather recipes how to get to these. Together with this recipe, one or more test suites would be provided that would check the integrity of the result.
All this would be build against the newest OS flavors or other dependencies. The build and test process would be performed automatically at regular intervals. In case of problems occurring at any step, expert intervention would be triggered in a timely manner for a single problem.
We will shortly discuss the advantages (+) and disadvantages(−) of such an approach, compared to a scenario in which software is not compiled and OS images are frozen. These issues are discussed in more details elsewhere[5].

– At first glance, the continuous rebuilding looks like a higher effort. It clearly is in the beginning, if the experiment does not already have an automated build and testing infrastructure. If there is, the task would be to maintain this infrastructure after the end of the active analysis period. An experiment-independent framework maintained elsewhere (e.g. in an IT department) would be helpful

– Some kind of organization would still be necessary even after the end of the collaboration: Experts needs to be named that are able to address potentially occurring problems.
+ Runability and correctness of the code is guaranteed at any given moment.
+ Changes between each test run are small if the frequency is high enough. This eases problem finding.
+ Necessary changes are potentially small.
+ As the software "lives", adaptations to new technologies (network, protocols, data analysis, MC generators, ...) can be made with reasonable effort.

2.2. Towards a DESY wide solution: A Generic Recipe

We have tried to identify a basic, generic recipe for all steps of the different HERA experiments– the atomic life–cycle of a validation test. This comprises six steps that involve either IT or experiments at some stage. Based on this, we have designed and implemented a framework in which such steps can be executed [6].

3. The Generic Recipe

3.1. Concept and Design

The prototype of our software preservation system is based on some general concepts. The modular design of the framework allows to replace components easily and to add new components to extend the functionality. New and unforeseen requirements by future environments may be included by minor changes of the structure of the code. Each configuration is formatted in plain text, human readable and self-explaining. This ensures longevity of the information both for operators and users. Open standards are preferred against proprietary ones. The communication between the components is mostly based on HTTP.

Data preservation covers a broad view of different fields. The focus is set rather on the software layer than on data archival or access. The standard *Open Archival Information System* (OAIS) [8] provides a general reference model for long–term preservation of digital data. It is identical to ISO 14721:2003. Our prototype is OAIS oriented but not OAIS conform as the standard has shortcomings with respect to software preservation and validation.

The validation system is composed of six different components as summarized in Fig. 3.1.

o The **controller** is the central component and manges all other components of our software preservation system. Incoming messages are processed and generate new events or tasks which, then, are transmitted to appropriate other components.

o The **ingest manager** executes all necessary steps needed to integrate a new test collection into the system. It extends a test specification with additional information, downloads external software packages and stores previous test collections.

o Most parts of the tests are executed inside virtual machines. The **workflow engine** provides mechanisms for mapping a life–cycle (see Fig. 3.1) inside a virtual machine.

o **IaaS cloud service.** The Infrastructure as a Service (IaaS) provider is responsible for managing and hosting virtual machines that can be instantiated dynamically and automatically for the software preservation system. Therefore, it is crucial to define a set of common interfaces being provided by the IaaS system. Providers of cloud computing differ from each other concerning the offered interfaces and protocols. Using only one of these interfaces may lead to a problematic dependency. In our prototype only open interfaces are used which are already standardized or at least well established such as the Open Cloud Computing Interface (OCCI[9]) driven by the Open Grid Forum[12]. OCCI is used as a protocol and as an API for any management tasks where the focus is set on portability, integration, innovation and interoperability.

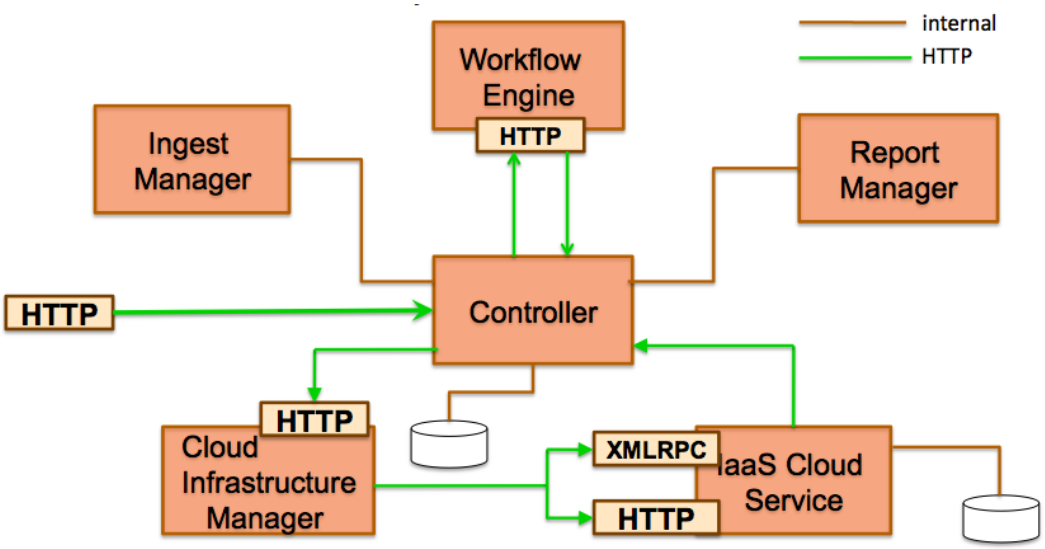

Figure 1. Components and their communication protocols.

- The **cloud infrastructure manager** comprises all feature related to IaaS interaction with the software preservation system. It is a management layer between controller and IaaS cloud service which decouples specific cloud protocols or function from the controller. This enables an exchange of the cloud service without touching the controller but requires only a modification in the cloud infrastructure manager itself.
- The **report manager** provides statistics and reports about test runs and the software preservation system itself. The output can be rendered as a JSON message or presented as HTML.

During the life–cycle of the system different types of resources such as virtual machines, software packages, scripts, system state values, log–files, test results, configuration files and test packages are created. Most of them are used by different components and during different time periods. Some need to be available only temporarily while others must be stored permanently. Our prototype uses Mercurial[11] as a source code management system and SQLite[10] as a database management system. The configuration data are stored in an ext3 file system.

A major pillar of the software preservation system is the integration of vitalization and cloud services to enable continuous testing of software packages. OpenNebula [7] is used for providing a cloud infrastructure as a service (IaaS). It can easily be adapted to special needs and is open–source.

Each operation accessing the cloud, for instance if a virtual machine image is requested, is handled at first by the controller. It receives messages and creates new actions. This may result in sending requests to the cloud–infrastructure manager. The cloud service is hidden from other components since only the controller is allowed to communicate with the cloud–infrastructure manager. OpenNebula can easily be replaced by a different cloud system, only the cloud–infrastructure manager needs to be modified.

3.2. Prototype Implementation

Each test collection consists of a set of configuration files. They are collected in a tar–gzip archive and sent to the software preservation system via a HTTP–POST request.

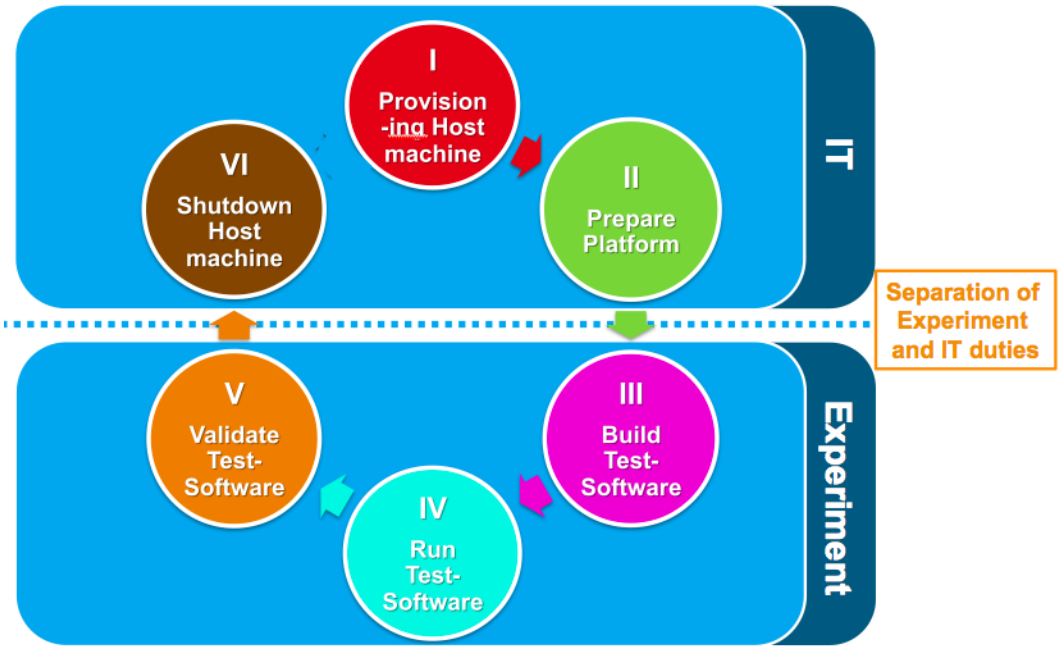

Figure 2. Life–cycle of a validation test. The responsibilities for the individual steps are separated between the IT group and the users.

The JavaScript Object Notation (JSON) is used as common notation for all configuration files. The basic idea is to use only key–value pairs, each element has the datatype string. By this structure it is not difficult to interpret the content in other programming languages. Additionally the key–value notation is easily extendable. New attributes can be added without affecting with the old ones.

- software.txt
  This file describes the analysis software to be validated by the software preservation system. The key *archive* characterizes a single archive that contains the full analysis software. If the software needs to be compiled the key builder describes the file that can be used for building the software. The key *executable* represents a script executing the analysis software. Both builder and executable have to be part of the compressed software archive.
- validator.txt
  Validator is a program that can validate results of an analysis software. The result and log files of an analysis software are passed to the validator which then returns the test result, for instance SUCCESS or FAIL.
- rpm.txt
  This file contains a list of key–value pairs describing the packages the analysis software is depending on. For example, the key *zip* may be the name of a package and its metadata are stored as a value. The package name must be available for the package manager which is installed in the virtual machine template. For each virtual machine package managers are defined in their metadata.
- vm.txt
  Details about the virtual machine used to run the analysis software on, are stored here. The software preservation system provides a list of virtual machine templates. Each template can be contextualized and instantiated.

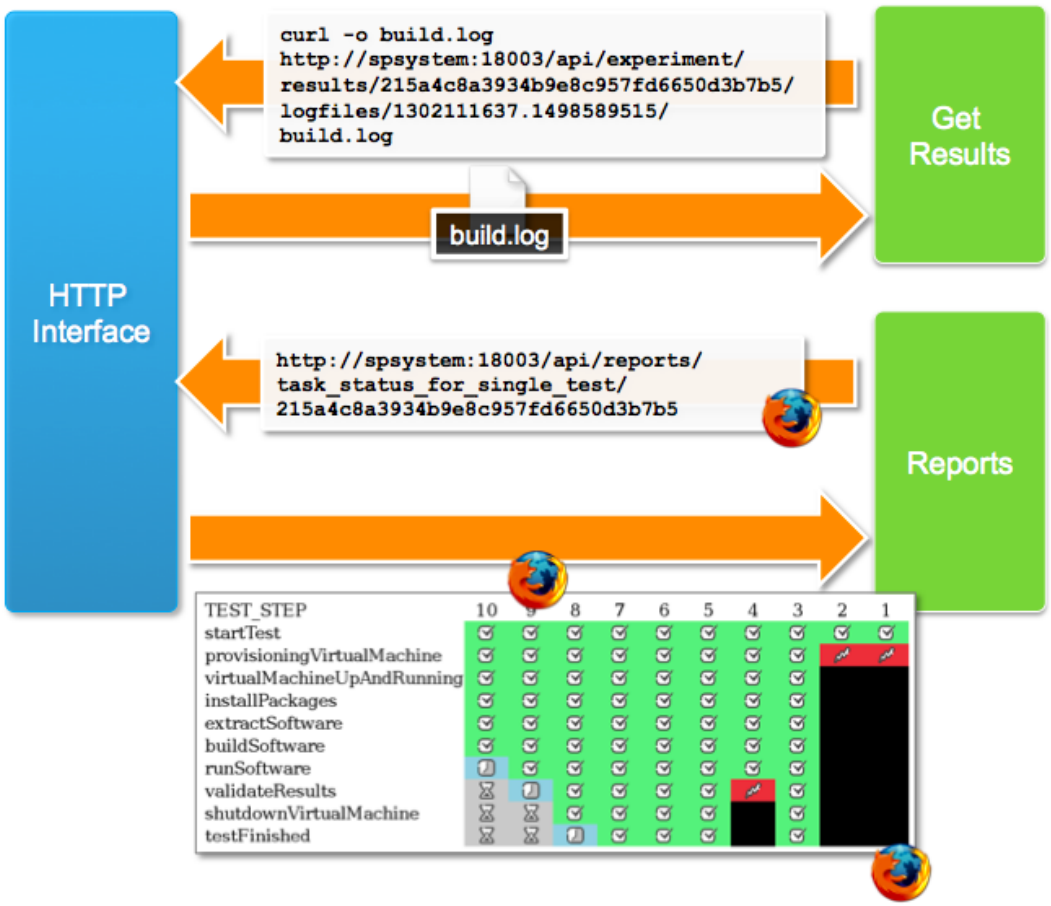

Figure 3. Monitoring.

- o contextualization.txt
  Each test has different requirements, therefore virtual machines must be prepared individually to fit the needs. In the current system this is achieved by the contextualization mechanisms of OpenNebula. The first time a virtual machines boots a shell script is executed to prepare the system (e.g. init.sh).
- o configuration.txt
  All remaining information about the test collection are part of this configuration file.

As soon as a valid archive is created it can be registered to the software preservation system by sending a HTTP–POST request to a webservice, for example via the command-line tool curl. Then, the software preservation system provides a reference number, the TESTCOLLECTION UUID. This id is then used to trigger a test run for the referenced test collection which will generate a new TESTRUN UUID.

The execution of each test step produces a log file. A user can download these files by sending an HTTP–GET request to the software preservation system, specify an id for the test and its test run as illustrated by Fig. 3.2. Furthermore the software preservation system provides graphical reports that can be rendered by a web browser as indicated at the bottom of Fig. 3.2.

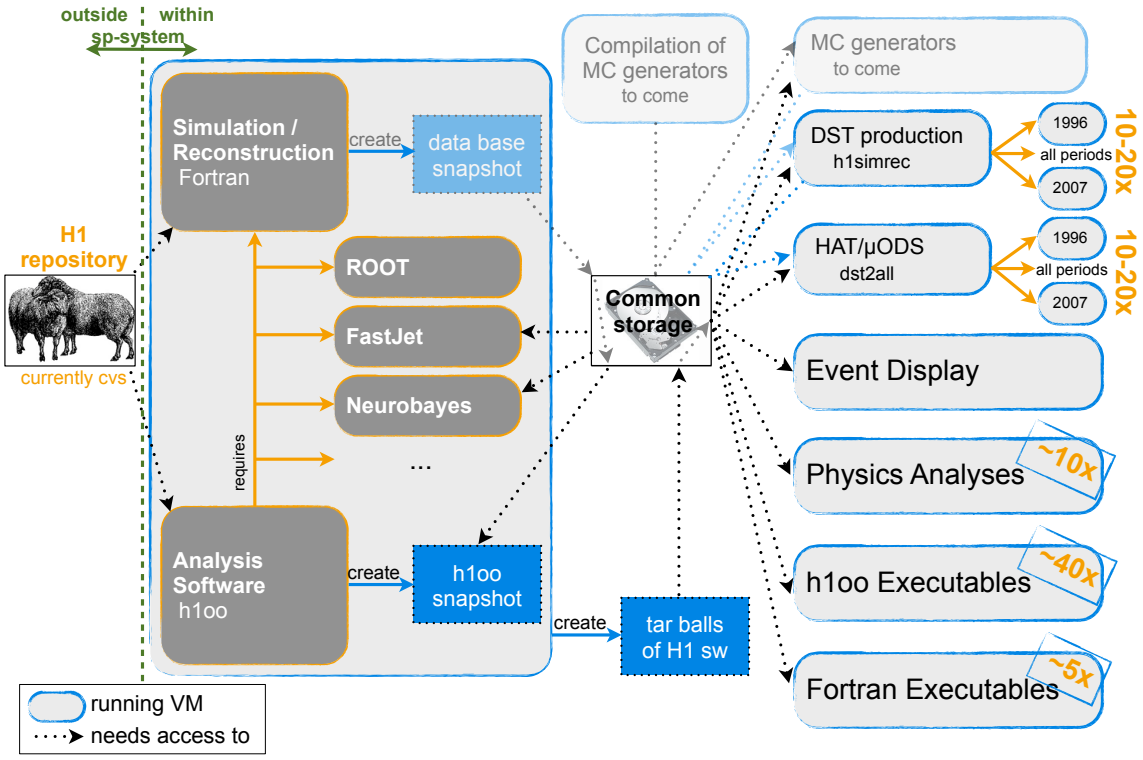

Figure 4. An example of complex workflows within the compilation, running and testing procedure and their interdependencies. (courtesy Michael Steder, H1).

4. Experiences with the Generic Recipe and the prototype implementation

HERA users have generally well received the idea of keeping all or parts of their code alive. Some concern has been raised that migrating to newer operating systems might be difficult or impossible in the future. The framework could also be used to test binaries compiled on an old OS which are executed in an new OS. This is sometimes possible to some extend using compatibility libraries.

Users have started writing build, run and evaluation scripts in parallel to the prototype implementation. At a rather late stage of the implementation, these scripts were run on the system. We identify two main lessons:

(i) Users tend to recreate the environment they are used elsewhere. It seems to be more work to disentangle the dependencies against e.g. AFS and software being available there then to recreate such environments. We think that for the long term success of the system, users must be helped in understanding such dependencies and solving them in a clear way.

(ii) It has turned out that the workflows of the experiments are more complicated than initially thought and need information exchange between different test scenarios. It is for example envisageable to separate the compilation of the whole framework into smaller pieces (e.g. modules) which are also tested independently. Some modules might however depend on the compilation of previous modules, so the modules from previous compilations should be available in the framework. Figure 4 shows a sketch of the H1 plans for such a structure.

This was not foreseen in the initial design. Any implementation that would go into real production should be able to handle such scenarios.

5. Conclusion and Outlook

In this contribution, we discuss different scenarios of preserving the ability to analyze data.
We have successfully planned and implemented a proof of concept of one scenario, which allows physicists to constantly and automatically validate their code and help migrating to new OS or hardware.
Experience of HERA physicists working with this prototype has shown that it is useful to some extend for them. It has turned out that their workflows are more complex that initially thought. All of the steps – compilation, running, testing – should be divided into smaller sub–steps with inter–dependencies.
To achieve production readiness, more resources and work are needed to plan and implement a framework which is aware of such complex workflows and inter–dependencies.

References

[1] ICFA Study Group on Data Preservation and Long Term Analysis in High Energy Physics: `http://www.dphep.org`
[2] DPHEP Study Group: Data Preservation in High Energy Physics: arXiv:0912.0255 [hep-ex] (2009).
[3] South, David: Data Preservation in High Energy Physics, CHEP 2010, arXiv:1101.3186v1 [hep-ex]
[4] Neal, Homer and Cartaro, Tina: Data Preservation at BaBar/SLAC, Presentations at the 5th DPHEP Workshop, FNAL 2011. Conference page: `http://indico.cern.ch/conferenceDisplay.py?confId=116485`
[5] Kemp, Yves: Use of Virtualization Techniques for data preservation and long term analysis. DPHEP workshop DESY 2009.
`http://www.dphep.org/sites/site_dphep/content/e20/e36625/e43750/dlpta.pdf`
[6] Strutz, Marco: Virtualisation Technologies and Cloud Computing in Data Preservation for High Energy Physics. Master's thesis, Berlin University of Applied Sciences (HTW), Berlin (2011).
[7] OpenNebula: Open Source Toolkit for Cloud Computing. `http://opennebula.org`
[8] Iso 14721:2003, Space data and information transfer systems - open archival information system - reference model. OAIS, 2003.
[9] OCCI, Open Cloud Computing Interface. `http://occi-wg.org/`
[10] SQlite: A SQL database engine. `http://www.sqlite.org/`
[11] Mercurial - A Distributed Source Control Management Tool. `http://mercurial.selenic.com/`
[12] OGF - Open Grid Forum. `http://www.ogf.org/`