Recording the LHCb data and software dependencies

Ana Trisovic[1,2], Ben Couturier[1], Val Gibson[2] and Chris Jones[2]

[1] CERN, Geneva, Switzerland

[2] University of Cambridge, Cambridge, the United Kingdom

E-mail: ana.trisovic@cern.ch

Abstract.

In recent years awareness of the importance of preserving the experimental data and scientific software at CERN has been rising. To support this effort, we are presenting a novel approach to structure dependencies of the LHCb data and software to make it more accessible in the long-term future. In this paper, we detail the implementation of a graph database of these dependencies. We list the implications that can be deduced from the graph mining (such as a search for the legacy software), with emphasis on data preservation. Furthermore, we introduce a methodology of recreating the LHCb data, thus supporting reproducible research and data stewardship. Finally, we describe how this information is made available to the users on a web portal that promotes data and analysis preservation and good practise with analysis documentation.

Keywords— data preservation, data provenance, graph database

1. Introduction

The Large Hadron Collider beauty (LHCb) experiment at CERN is a general purpose detector in the forward region, which focuses on investigating the differences between matter and antimatter by studying the decays of beauty (B) and charm (D) mesons. The detector has been recording data from proton-proton collisions since 2010 and is expected to record data throughout the 2020s. Due to the rapid development of both the hardware and software used to process the data, many questions have been raised about data compatibility and preservation. In response to this, we are creating a database to record the metadata of our software and the data provenance. The recorded dependencies are expected to ease the process of running the software and analysing the data in the long-term future.

1.1. Data preservation initiative

The data preservation project in High Energy Physics (HEP) aims to ensure the preservation of experimental and simulated data, as well as the scientific software and documentation. The main objective is to assist analysing HEP data in the future. The major use cases include looking for signals predicted by new theories and improving current measurements, in addition to physics outreach and educational purposes.

The LHCb data is processed with software and hardware that are changing over time. The information about the data, software and the changes have been logged in the internal databases

and web portals. Our goal was to collect and structure this information into a singular, robust database that can be used immediately for scientific purposes and for the long-term future preservation.

Finally, in support of making the LHCb data replicable and reusable, we concentrate our efforts to provide complete information on the data provenance and, to some extent, assist in its recreation independently of the CERN infrastructure.

2. Overview of the LHCb software and data production

The LHCb data production consists of the recording of the data by the LHCb detector, the processing of this data, and finally making the data available to physicists. In HEP analyses we also use the result of Monte Carlo simulations, which represent simulations of particle collisions. It is commonly used to help to characterize the signal (particle decay) of interest. These samples are kept in the same data format as the experimental (real) data and handled with the same software. In this paper, the term *data* will be used to refer to both simulated data and the real experimental data.

The LHCb software is based on the Gaudi framework [1]. Gaudi is an object-oriented framework designed to provide a common infrastructure and environment for the different software applications of the LHCb experiment [2]. It is a modular software, which supports event data processing in real time inside the experiment, the data production in the offline system and the physics analyses performed by the users.

In each stage of the data production, we use a specialised software application. There are a number of versions of each application, in some cases over 100 different versions to date. Different versions can produce different physics output, thus making our data and physics results biased. Therefore it is crucial to know how the application manipulates the data in use.

Initial stages of the Monte Carlo production [3]:

- **Simulation - Event generation.** The application *Gauss* mimics what happens inside the spectrometer of the LHCb experiment, simulating the initial data set of particle collisions [2].
- **Digitization - Detector response.** The application *Boole* performs the simulation of the detector responses to produce the output in the same form as the experiment's electronics.
- **L0 trigger emulation and High level trigger (HLT).** Only a small fraction of the events produced in the LHCb detector are interesting to analyse. The trigger is a mechanism to select these events and save them, discarding the rest of the data (around 99% of what was produced). The application that runs the triggers is *Moore*. Moore filters the simulated data in the same way as real data.

The final stages in data production are the same for Monte Carlo samples as for the real data obtained by the data acquisition (DAQ) system (shown in Figures 1 and 2). The settings in the applications are equivalent for both real data and simulation, to keep one consistent with the other:

- **Turbo.** Turbo is a new streaming strategy where events are reconstructed in the trigger, thus bypassing the offline reconstruction and discarding the raw events [4]. The application used in turbo stream, *Tesla*, writes out a compact summary of physics objects containing all the information necessary for analyses.
- **Reconstruction.** In this stage, the detector hits are transformed into physics objects and written to disk. The *Brunel* application integrates the complete pattern recognition and

produces files containing all reconstructed items such as calorimeter and trackers clusters, charged tracks and information on particle identification.

- **Streaming.** Streaming is the final stage of the production, which covers selection of the events. This is done with the *DaVinci* application [2].

Figure 1 shows the real data production pipeline. After data acquisition in the *Online* system, the data is stored in RAW data format. It is further processed in *Reconstruction* and *Streaming* or *Turbo* stages, before it is stored on a disk and made available to the physicists. Figure 2 shows Monte Carlo simulation pipeline. The dice in the beginning of the pipeline represent a random generator, that is a central part of the simulation generation. The names of stages are followed by the names of applications used in each stage.

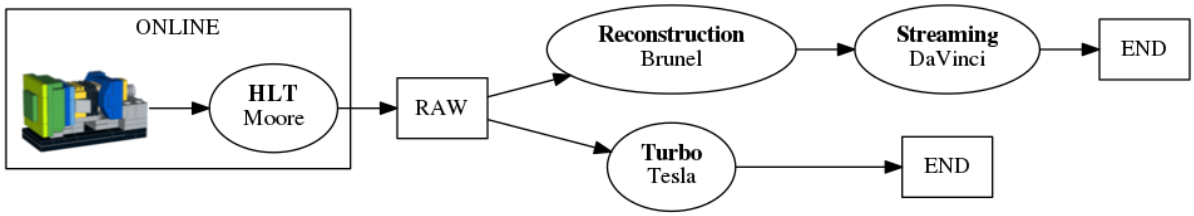

Figure 1. The real data production pipeline

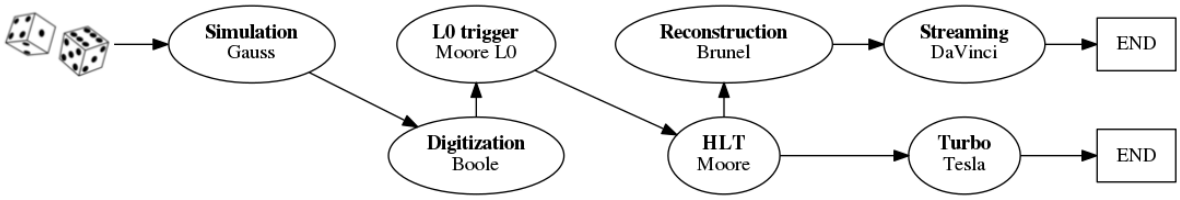

Figure 2. The simulation production pipeline

3. Implementation

3.1. The database design and technologies

For the storage implementation we chose a graph database, as it provides flexibility in object dependencies that relational databases cannot capture. Firstly, the modules in the LHCb software are connected in a complex way that can be elegantly presented in a graph. Secondly, by adopting a graph database approach, we are able to accommodate the changes that come with time and the new LHC run standards. For instance, we can easily add to the nodes a new attribute for *data quality*, even though they were not there originally.

A graph database is a storage engine that stores data as nodes and relationships and allows high performance retrieval and querying of those structures. Properties can be added to both nodes and relationships. One of the most prominent graph databases, and the technology used in our solution, is *Neo4j*[1]. It is an open source NOSQL graph database implemented in Java.

[1] Official web page of the Neo4j technology: https://neo4j.com/

Objects in *Neo4j* are manipulated with the *CYPHER* query language, which is SQL-inspired, declarative and can describe patterns in graphs using an ascii-art syntax.

Our database implements the following nodes:

- **Production.** A production node captures a list of steps which define the LHCb data production pipeline. There are three types of *Production* nodes, and those are: simulation models (prerequisite for simulation production), simulation and real data production.
  Every node is described with a list of attributes (as shown below in JSON). The *Name* value includes the year of data-taking (here 2016), beam energy and the kind of collision, in this case that is *Proton-Argon*. At LHCb we mostly examine proton-proton collisions, but also proton-helium, proton-neon etc. *ID* stands for the identification number of the production and *DQflag* is a data quality flag. Every step has its own ID number and a list of configuration tags, such as *DDDB* and *CondDB* tags which define detector and conditions respectively, python *Options* file and *Input* and *Output* data formats.

```
{
  "ID": "31034",
  "Name": "Reco16-Beam2510GeV-MagDown-ProtonArgon",
  "Type": "Reconstruction",
  "EventType": "90000000 Full stream",
  "ProcessingPass": "Real Data",
  "DQflag": "OK",
  "Step": [
    {
      "ID": "129609",
      "Application": "Brunel-v50r1",
      "System_config": "x86_64-slc6-gcc49-opt",
      "Options": "$APPCONFIGOPTS/Brunel/DataType-2015.py;$APPCONF...",
      "DDDB": "dddb-20150724",
      "CondDB": "cond-20160517",
      "Extra": "AppConfig.v3r272",
      "Input": "RAW(Y)",
      "Output": "BRUNELHIST(Y),FULL.DST(Y)"
    },
    { ... }
  ]
}
```

- **Project.** A project describes one version of a module in the LHCb software stack. The class is defined with a project name and version, for example *BRUNEL v44r8* or *DAVINCI v36r2*. The projects are built depending on each other, and each project version requires specific set of other projects to run. To demonstrate this *REQUIRES* relationship, below is the *CYPHER* code that finds a path ***p*** from an application in the top level of the LHCb software stack DaVinci, to the Gaudi framework, which is the base of the stack. The result of this query is a path which normally contains several nodes and includes all the projects that are required to run the DaVinci application.

```
MATCH   p = (a:Application{project:'DAVINCI'})-
            [r:REQUIRES*..]->
            (d:Framework{project:'GAUDI'})
            RETURN p LIMIT 1
```

An example of the output of the previous query is:

`DAVINCI v33r1 → ANALYSIS v10r3 → PHYS v16r3 → REC v14r3 → LHCB v35r3 → GAUDI v23r5`

- **Platform.** A platform describes a computer environment that supports the LHCb software. For instance, `x86_64-slc6-gcc49-opt` means optimized (opt) `x86_64` architecture of Scientific Linux CERN 6, and conventionally gcc49 stands for GNU Compiler Collection (GCC). Links between *Projects* and *Platforms* define compatible platforms for each project. For example, a list of compatible platforms for *DAVINCI v38r0* is shown in the table below.

Project	Platform
DAVINCI v38r0	`x86_64-slc6-gcc49-opt`
	`x86_64-slc6-gcc48-opt`
	`x86_64-slc6-gcc48-do0`
	`x86_64-slc6-gcc49-dbg`
	`x86_64-slc6-gcc48-dbg`

Object-Relational Mapping (ORM) in computer science is a programming technique for querying and manipulating data from a database using an object-oriented paradigm. For each node label in the graph database (*Production, Project, Platform*), there is a corresponding Python class, which enables the use of the objects from within the programming language. The ORM used in this project is called *neomodel*[2] and it is specific for the *Neo4j* graph database.

An example to illustrate the schema of the database is shown in Figure 3.

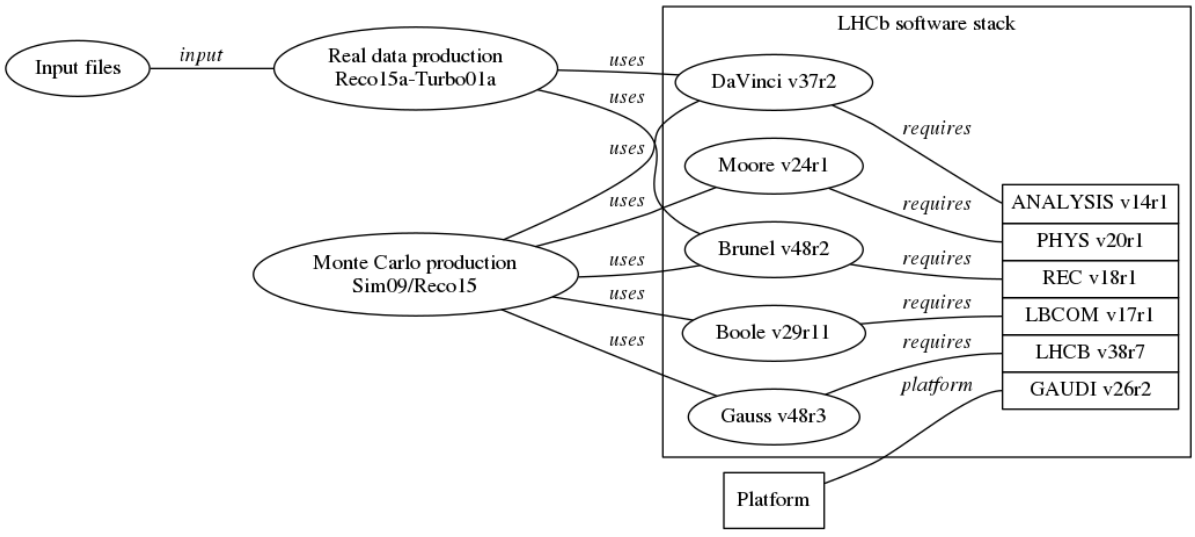

Figure 3. The graph database schema.

3.2. Application Programming Interface API

The API to the LHCb graph database is implemented in *py2neo* which is a client library and toolkit for working with *Neo4j* from within Python applications and from the command line. We use the Representational State Transfer (REST) [5] architectural style to create a

[2] More about *neomodel* can be found on GitHub: https://github.com/robinedwards/neomodel

request/response mechanism between the server and the client. The data retrieved directly from the database is returned in JSON format. Some of the functions are listed below:

- `getProductionSequence(ID)` Returns a *Production* entity with the given *ID* number.
- `listProjects()` Returns the list of *Projects*.
- `listPlatforms(Project)` Lists the suitable *Platforms* for the given *Project*.
- `listRequirements(Project)` Lists the required modules for the given *Project*.

3.3. Web portal

The graph database is connected to a *Django* web server and made available for the users as a web portal. Django is a high-level, open source Python Web framework that encourages clean and pragmatic design of the web applications [6].

The web portal allows browsing through the LHCb data production and software information. It implements a search engine for this metadata, and it is meant to recommend the latest streaming to the users. In addition, it should promote the LHCb data preservation and provide a comprehensive user guide for analysis documentation and preservation.

4. Functionality

4.1. Methodology for preservation and recreation

The instances of the class *Production* consist of a sequence of steps, where every *Step* provides complete configuration information to rerun one stage in the data production. This configuration information is easily transformed into a *Python file* that can be run in the LHCb computing environment, thus producing the data. However, the production can be run outside the CERN infrastructure, making the LHCb data replicable and sustainable for the long-term future. Next to the configuration information, other requirements include:

- **Software from CERN Virtual Machine File system (CVMFS)**[3]**.** CVMFS is an open source file system designed for efficient software distribution [7]. The HEP experiments install all released software components in its final configuration on CVMFS [8].
- **Computing environment.** The LHCb software is developed for the Linux operating system, specifically Scientific Linux CERN 5 or 6 (slc5 or slc6 respectively). If these distributions are unavailable, it is necessary to mount an image on virtual machine or container engine such as *Docker*[4] to run Linux and the LHCb computing environment from CVMFS.
- **Input files (RAW).** In real data processing, input files created at the LHCb detector are required. These files need to be stored in such way to be accessible from within the application. In Monte Carlo production, input files are not required as all the events are simulated inside the application.

4.2. Recording data provenance

Data provenance has evolved to be an integral part of best practise for long-term data preservation and research reproducibility. It certifies the origin of the data, and in computational terminology it is defined as the representation of the relationship between data items (*entities*),

[3] Official web page of CERN VMFS: https://cernvm.cern.ch/portal/filesystem
[4] Official web page of the Docker technology: https://www.docker.com/

transformations (*activities*), and individuals or organisations (*agents*) in a graph. These graphs capture when, by whom and how a data item was created and/or processed. Provenance systems typically concern some aspects of: data quality, replication recipes, ownership attribution, context understanding and audit [9].

The LHCb graph database records the data provenance by explicitly linking the raw data sets to processing sequence they went through. It mostly follows the typical structure, except for the absence of the executor, as the activities are organized by the LHCb experiment collaboration. The LHCb software is open source and available on GitLab[5], which allows the users to independently track the computational performance.

4.3. Source for CERN Analysis Preservation portal

The CERN Analysis Preservation (CAP) portal is a joint project across the LHC experiments at CERN [10]. They manage the computing resources and a portal for analysis preservation. The aim is to promote good practise in performing physics analysis and assist in analysis preservation and reproducibility.

On the web portal, analyses should be documented in a form completed by the authors. The portal can query the working group databases and the graph database to get some of the information needed in the form. The form sends a REST request with parameters and the server returns required information in JSON. Currently, the API provides the application used in a given production, and platform information for each application.

4.4. Graph mining

Graph mining is the process of finding and extracting concealed information from graphs. This information can be worthwhile for the future project management and the community, particularly for the long-term preservation activities. The information that we can easily extract from the LHCb graph database are:

- **The most used project versions.** The *Project* nodes with the highest degree in the graph are the most used versions. They are likely to be often reused in future and should be preserved having this in mind.
- **Obsolete project versions.** On the contrary, we can identify obsolete software versions as the *Project* nodes with the lowest degree. This information is valuable because we can archive and remove them in a new release of CVMFS.
- **Project classification.** We can classify the projects into groups where each of them run with a specific subset of the data (eg. 2011 data). This is particularly important for testing the projects against their corresponding data sets and verifying they work in the same way in the future as they work today.
- **Troubleshooting.** Identifying the data sets handled by a faulty software component. Given that there is a defect in one of the projects, we can search through the graph to find the affected data sets.

4.5. Other use-cases

The graph database is regularly used in the LHCb software production, providing the information on project (module) dependencies and the order in which they should be built. Other use-cases

[5] GitLab repository of the LHCb software: https://gitlab.cern.ch/lhcb-core

include: recommending the latest streaming, searching and browsing through the LHCb software and data tags, finding all the modules required by a project and vice versa, finding the modules supported by a project, eg. what project are run on top of *GAUDI vXrY*. This functionality is provided to the users at the web portal.

5. Conclusion

In this paper we presented the LHCb graph database, a novel approach for recording dependencies between the software and data. The data nodes in the graph represent each production of experimental or simulated data, while the software nodes represent each project within LHCb, how it depends on other projects (modules) and how it is linked to the data production. Furthermore, we introduced a methodology to recreate a data set independently of the CERN computing infrastructure using the information stored in the *Production* nodes. We are concentrating our efforts to advance this methodology to make the LHCb data replicable, reusable and sustainable for the long-term preservation.

Acknowledgements

The authors would like to thank Marco Cattaneo and Gloria Corti for providing inside information about the data and simulation production at LHCb. We are grateful for their constructive suggestions for the data preservation initiative and for answering our questions. Thank you to Philippe Charpentier, Federico Stagni and Zoltan Mathe for answering our questions about LHCb Bookkeeping and Dirac. Special thanks to Vladimir Gligorov and Agnieszka Dziurda for following this project and providing their insights and ideas. Thank you to Silvia Amerio for suggesting various use-cases. Thank you to Illya Shapoval for starting off the use of Neo4j at LHCb.

References

[1] Barrand G, Belyaev I, Binko P, Cattaneo M, Chytracek R, Corti G, Frank M, Gracia G, Harvey J, Van Herwijnen E *et al.* 2001 *Computer Physics Communications* **140** 45–55

[2] Corti G, Cattaneo M, Charpentier P, Frank M, Koppenburg P, Mato P, Ranjard F, Roiser S, Belyaev I and Barrand G 2006 *IEEE transactions on nuclear science* **53** 1323–1328

[3] Corti G, Charpentier P, Clemencic M, Closier J, Couturier B, Kreps M, Màthè Z, O'Hanlon D, Robbe P, Romanovsky V *et al.* 2015 How the monte carlo production of a wide variety of different samples is centrally handled in the lhcb experiment *Journal of Physics: Conference Series* vol 664 (IOP Publishing) p 072014

[4] Benson S, Gligorov V, Vesterinen M A and Williams J M 2015 The lhcb turbo stream *Journal of Physics: Conference Series* vol 664 (IOP Publishing) p 082004

[5] Jakl M 2005 Representational state transfer

[6] Holovaty A and Kaplan-Moss J 2009 *The definitive guide to Django: Web development done right* (Apress)

[7] Buncic P, Sanchez C A, Blomer J, Franco L, Harutyunian A, Mato P and Yao Y 2010 Cernvm–a virtual software appliance for lhc applications *Journal of Physics: Conference Series* vol 219 (IOP Publishing) p 042003

[8] Shiers J, Berghaus F O, Cancio Melia G, Blomer J, Dallmeier-Tiessen S, Ganis G and Simko T 2016 Cern services for long term data preservation Tech. rep.

[9] Simmhan Y L, Plale B and Gannon D 2005 *ACM Sigmod Record* **34** 31–36

[10] Dallmeier Tiessen S, Herterich P, Igo-Kemenes P, Šimko T and Smith T Cern analysis preservation (cap)-use cases (2015)