Workflow Management for Complex HEP Analyses

To cite this article: M Erdmann et al 2017 J. Phys.: Conf. Ser. 898 092043

View the article online for updates and enhancements.
Workflow Management for Complex HEP Analyses

M. Erdmann, R. Fischer, M. Rieger, R. F. von Cube
III. Physics Institute A, RWTH Aachen University, Germany
E-mail: rfischer@physik.rwth-aachen.de

Abstract. We present the novel Analysis Workflow Management (AWM) that provides users with the tools and competences of professional large scale workflow systems, e.g. Apache’s Airavata[1]. The approach presents a paradigm shift from executing parts of the analysis to defining the analysis. Within AWM an analysis consists of steps. For example, a step defines to run a certain executable for multiple files of an input data collection. Each call to the executable for one of those input files can be submitted to the desired run location, which could be the local computer or a remote batch system. An integrated software manager enables automated user installation of dependencies in the working directory at the run location. Each execution of a step item creates one report for bookkeeping purposes containing error codes and output data or file references. Required files, e.g. created by previous steps, are retrieved automatically. Since data storage and run locations are exchangeable from the steps perspective, computing resources can be used opportunistically. A visualization of the workflow as a graph of the steps in the web browser provides a high-level view on the analysis. The workflow system is developed and tested alongside of a ttbb cross section measurement where, for instance, the event selection is represented by one step and a Bayesian statistical inference is performed by another. The clear interface and dependencies between steps enables a make-like execution of the whole analysis.

1. Motivation
Modern high-energy physics data analyses are growing in complexity and scale. Adding to the complexity are for instance the number of advanced analysis techniques like multivariate classifiers, sometimes requiring graphics processing units, the number of simultaneous analyses, e.g. for various final state channels or event categories. The scale increases as more data get recorded, and more simulated events are needed to describe the data with proper statistics. Issues of scale are typically addressed via high throughput computing concepts involving a large number of computing jobs and many computing sites, e.g. via the Worldwide LHC Computing Grid (WLCG)[2]. The user is presented with the task to execute the analysis under these conditions. This involves a lot of bookkeeping to make sure that each required file is at the right place at the right time. Due to the complexity, often only the physicist knows the exact order in which certain scripts need to be executed or which files need to be copied where to perform the analysis. New team members or students are often challenged to learn many technical details on how to run the analysis. Finally, if people leave a group, they often take key knowledge with them. The focus of analysis documentation in publications or conference talks is mostly on what was performed, not on how it was done on a technical level. These factors contribute to making it difficult to extend or replicate an existing analysis.

The Analysis Workflow Management (AWM) system aims at changing the analysis paradigm from executing to defining the analysis. If the user defines what is the input data, what does
the input depend on, and how it should be processed, the system should be able to execute the task. The goal is to run the analysis make-like [3] in a distributed manner.

2. Analysis Step
A data analysis comprises certain steps. Typical examples of analysis steps in a high-energy physics are event selection, training of classifiers, statistical inference, plotting, or auxiliary estimations of scale factors or other relevant information. Often one part of the analysis consists of a number of steps. An example could be a data driven background estimation that first needs to create data tuples with relevant information, then performs a fit in a sideband, and subsequently creates control plots.

AWM represents the analysis in these natural steps. In principle each step is an operation on its input data and additional parameters, and produces output data. Often the input data comes in the form a list of file names. The step is then supposed to iteratively process each of the input files. The payload, which does the actual work to be performed for each iteration of the step’s execution, can be any executable. Figure 1 shows an illustration of the analysis step.

AWM is implemented in Python and acts as the fabric to connect multiple parts of the analysis with each other. Since the focus is on analysis steering, there are no limits to which programming languages or data formats can be used.

3. Data Collections
In order to create dependencies between multiple analysis steps such that one step’s output can be used as another step’s input, a clear interface between the steps is required. This interface is meant to give the maximum flexibility to the user. Therefore, it does not enforce a certain data format for data files. In AWM, the interface is realized via Data Collections. Data Collections are lists of literal items, i.e. strings, integers, floating point numbers, associated with properties that are implemented as a key value storage. Often step’s produce output files, in which case the step would produce a special type of collection suitable for files. File Data Collections store a list of references to files and have associated storage locations, as described in section 4.2. Often it is useful to attach small pieces of information with output data, e.g. a number specifying the selection efficiency used to generate the data. These are conveniently attached and retrieved via the properties of Data Collection items.

4. Abstraction of Computing Resources
In high-energy physics, data analyses are often executed opportunistically on various computers. This could be local computers, university batch systems, the computing grid, or other systems. AWM abstracts computing resources such that the user can switch Run and Storage Locations comfortably. Because the software environments varies from one Run Location to another, a mechanism to install required software is provided via the AWM Software Manager. Due to
the flexibility of AWM, steps may have different associated computing resources. The Run Conditions determine during scheduling of a step execution which resources are to be used and performs basic sanity checks. They ensure for example that a remote job does not try to access data that is only locally available on the user’s computer. For testing or debugging purposes, the user can at run time decide to override decisions regarding Run and Storage Locations that were made during the workflow configuration.

4.1. Run Locations

Run Locations provide an interface between AWM and submission systems. AWM has mechanisms to provide necessary configuration information on the steps and to create file bundles that need to be available for the execution at the Run Location. AWM supports local computers, HTCondor clusters [4], and the WLCG out of the box. In order to add a custom Run Location, a class can be derived in which the user needs to implement methods to submit jobs, check their status, and to retrieve the standard output and error files.

4.2. Storage Locations

Internally, File Data Collections as described in section 3 use logical file names to refer to a certain file. The actual location of the file varies for different Storage Locations. The trivial case is the local Storage Location in which case all logical file names are understood relative to a common parent directory. In order to run analyses in a distributed manner, other storage solutions are needed. The grid Storage Location provides access to files via the SRM protocol[5], Grid FTP, and the dcap protocol[6]. Custom storage location, e.g. for FTP or SSH can be implemented by providing methods to copy files to and from the corresponding storage, methods to access meta-data, e.g. file size or checksums, and to delete files.

Because all data files in AWM are accessed via the Storage Locations interface, it does not matter where the file is actually stored as long as it can be accessed from the current machine. Multiple storage locations can be associated to Data Collections to provide either redundancy or local file caching.

```python
py27Package = SoftwarePackage(name="python", version="2.7.6",
    sourceUrl="http://www.python.org/ftp/python/2.7.6/Python-2.7.6.tgz")
py27Package.addRuntimeDependency(pkgConfigPackage)
py27Package.setBuildScript(""
    cd {source_directory}
    mkdir -p {install_prefix}/lib
    ./configure --enable-shared --prefix {install_prefix}
    make -j{num_cores}
    make install"
    )
py27Package.setPreexecuteScript(""
    addbin {install_prefix_variable}/bin
    "")
py27Package.setTestScript(""
    if [ "xpython --version 2>&1" == "xPython {version}" ]; then
        exit 0
    fi
    exit 1"
    )
```

**Figure 2.** Code snippet with an example of a software package definition.
4.3. **Software Manager**

No matter where an instance of an analysis step is executed, the payload executable and its dependencies need to be available. Therefore, AWM allows the user to associate setup scripts with each Run Location and step in order to configure and use pre-installed software. However, if specialized or custom software is needed, this is not always sufficient. In case specific software packages are needed, the user can create software package descriptions. They contain a URL to a compressed file, e.g. containing the source code of the software, or to a SVN[7], GIT[8] or Mercurial[9] software repository. In addition to the URL, small setup and install scripts need to be provided. The user can define dependencies between software packages. Figure 2 contains a code snippet for a software package that uses the typical `configure`, `make`, `make install` scheme. The Software Manager will download the software package, run the compile and setup scripts on the Run Location using available compilers and libraries. To limit the number of downloads and not to waste computing time to compile from source each time, the pre-compiled software packages are compressed and stored on a Storage Location in binary form. Next time this software package is required under the same conditions, i.e. same Run Location and setup scripts, the pre-compiled binary archive is retrieved and installed.

In case software cannot be downloaded, either because it is not available online or because it is under current development and only up-to-date on the user’s computer, e.g. analysis code, software archives can be transferred to the Run Location as sandbox packages. Other than that they do not need to be downloaded later on; they are treated in a similar way as other software packages.

5. **Execution**

The execution scheme is shown in Figure 3. Each step is executed in one or multiple tasks. The Scheduler is the component that determines which tasks are unfinished. Unfinished tasks are sent to the Run Location configured for this step. Multiple tasks can be combined into one computing job if the user reckons the execution duration for one task is too short for one job. Once the job starts at the Run Location, the software environment is created using setup scripts and the Software Manager. After the environment is set up, the required input data is downloaded. If all required software and files are available the payload is executed. After a task is finished, the step stores metadata and provenance information in JSON [10] task reports on a Storage Location for meta data.

The next time the Scheduler identifies unfinished tasks it first downloads newly created task reports. Reports contain information on whether a task was performed successfully and if not what was the cause of the error. This allows to resubmit unsuccessful tasks already while the computing job is still running on other tasks. They also contain information on output Data Collections that were created including information on Storage Locations of output files. Subsequent steps can use these collections as their input Data Collection such that the Scheduler can schedule the execution of appropriate tasks for the subsequent step.

Because steps can spawn tens of thousands of tasks, reports are summarized into one file. Thereby, each task report only has to be read once. An example for useful metadata that can be evaluated by subsequent steps is the number of originally generated events for samples containing simulated events. This number can be used later in plot steps to calculate event weights and to scale the samples appropriately.

The user can specify whether dependent steps need to be finished entirely or whether it can already work with parts of the output data. For example, not all tasks of an event selection step have to be done for a following reconstruction step to start running on parts of the data. A step performing a statistical inference on the other hand might need all available reconstructed data to produce proper results.
6. Conclusion
The Analysis Workflow Management system presented here is well equipped to cope with the challenges of modern high-energy physics data analyses. It was developed alongside a measurement of the top quark pair associated bottom quark pair production at the LHC. Therefore all components are intensively tested. Workflow tools are able to help the user to focus on the actual data analysis. Bookkeeping tasks and undocumented states of files and intermediate steps are prevented. Workflow systems can further help to have multiple people working on the same analysis by sharing the same code and data files transparently. It is therefore easier to extend and replicate existing analyses.

Acknowledgments
This work is supported by the Ministerium für Wissenschaft und Forschung, Nordrhein-Westfalen, the Bundesministerium für Bildung und Forschung (BMBF), the Helmholz Alliance Physics at the Terascale, and the Deutsche Forschungsgemeinschaft (DFG).

References
[1] The Apache Software Foundation Apache airavata URL http://airavata.apache.org
[2] Bird I 2011 Annual Review of Nuclear and Particle Science 61 99–118
[3] Free Software Foundation, Inc 2016 GNU Make URL https://www.gnu.org/software/make
[4] Fajardo E M, Dost J M, Holzman B et al. 2015 Journal of Physics: Conference Series 664 062014
[5] Sim A et al. 2009 SRM v2.2 Specification Tech. rep. Lawrence Berkeley National Laboratory
[6] Fuhrmann P, Behrmann G, Bernardt C et al. 2012 dCache, agile adoption of storage technology (Computing in High Energy and Nuclear Physics, New York City (USA), 21 May 2012 - 25 May 2012) URL https://bib-pubdb2.desy.de/record/140491
[7] Apache Subversion URL http://subversion.apache.org
[8] Git URL https://git-scm.com
[9] Mercurial URL https://www.mercurial-scm.org
[10] 2013 The JSON Data Interchange Format Tech. Rep. Standard ECMA-404 1st Edition / October 2013 ECMA URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf