Giving *Pandas* ROOT to chew on: experiences with the XENON1T Dark Matter experiment

D Remenska[1], C Tunnell[2], J Aalbers[2], S Verhoeven[1], J Maassen[1], J Templon[2]

[1]Netherlands eScience Center, Science Park 140, Amsterdam, The Netherlands
[2]National Institute for High Energy Physics (NIKHEF), Science Park 105, Amsterdam, The Netherlands

E-mail: d.remenska@esciencecenter.nl

Abstract. All articles *must* contain an abstract. This document describes the preparation of a conference paper to be published in *Journal of Physics: Conference Series* using LaTeX 2$_\varepsilon$ and the `jpconf.cls` class file. The abstract text should be formatted using 10 point font and indented 25 mm from the left margin. Leave 10 mm space after the abstract before you begin the main text of your article. The text of your article should start on the same page as the abstract. The abstract follows the addresses and should give readers concise information about the content of the article and indicate the main results obtained and conclusions drawn. As the abstract is not part of the text it should be complete in itself; no table numbers, figure numbers, references or displayed mathematical expressions should be included. It should be suitable for direct inclusion in abstracting services and should not normally exceed 200 words. The abstract should generally be restricted to a single paragraph. Since contemporary information-retrieval systems rely heavily on the content of titles and abstracts to identify relevant articles in literature searches, great care should be taken in constructing both.

1. Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2. The XENON Dark Matter experiment

Most matter in our Universe is "dark matter", yet it remains invisible to our instruments. What constitutes dark matter is one of the most intriguing questions in modern physics. The XENON dark matter experiment [1], installed underground at the Laboratori Nazionali del Gran Sasso (LNGS), aims to detect dark matter in the form interactions of Weakly Interacting Massive Particles (WIMPs) with xenon nuclei. The XENON100 [2] detector uses a total of 161 kg of pure liquid xenon (LXe) divided in cylindrical two-phase (gas/liquid) time projection chambers. Its successor, the XENON1T detector [3], recently deployed at LNGS, is realized by scaling up the existing XENON100 detector by a factor of 10 and reducing the background by a factor of 100. With the detector being able to distinguish between the WIMP and possible background radiation, the researchers hope to be able to prove the existence of a new subatomic particle, and determine its mass and likelihood of interaction with ordinary matter.

2.1. Computing model and data flow

In preparation for the XENON1T data acquisition, the envisaged new computing model is shown on Figure 1. The experiment data flow contains three parts: acquisition, processing, and analysis. The data acquisition system (DAQ), in proximity to the detector, acquires data at a rate of up to 2.4 Gbps, digitizes the PMT signals and transfers the digitized waveforms from the FADCs (Fast/Flash Analog-to-Digital Converters) to a local MongoDB storage. The acquisition stage produces cumulatively 1PB of raw BSON-based data files (binary serialization format) that must be saved to tape. The raw data is subsequently partially processed and reduced using highly optimized trigger routines. The Celery distributed task queue [4] is used for parallelization. At the end of acquisition, all of the raw data remains within a single-sited database at the LNGS site. The utilization of a cloud or grid infrastructure is foreseen in the future, for more efficient larger-scale processing. The Processor for Analyzing XENON (PAX, Fig. 2), developed fully in Python, is used for digital signal processing and other data processing on the XENON100/XENON1T raw data. This program is its own R&D in both measurement techniques and sustainable software development in the sciences. With a series of I/O and transformation plugins developed for intermediate processing steps, PAX reduces a single event from roughly 1MB of raw data to 50 KB of processed data.

The default PAX output format is a ROOT file containing the Event class. The processing stage will produce 50 TB of ROOT files, which the analysts must be able to process in an efficient manner. For this purpose, the Handy Analysis for XENON (HAX) software was prototyped

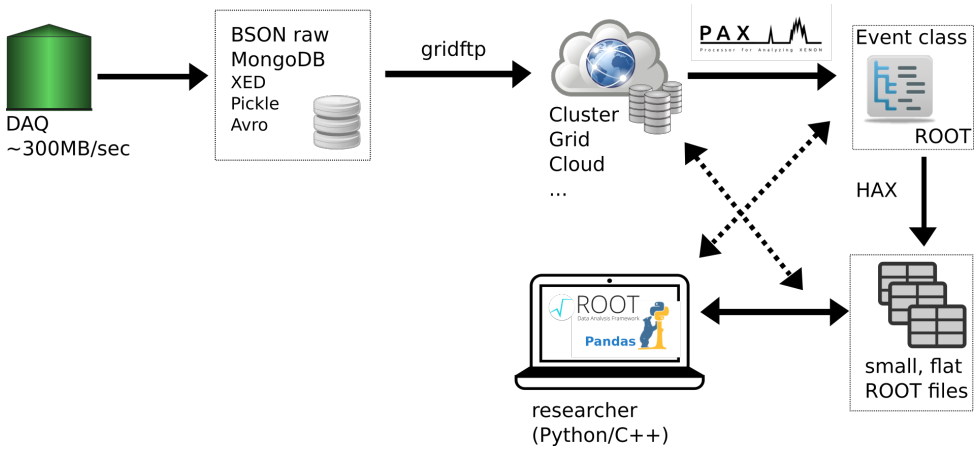

Figure 1: XENON1T computing model and data flow

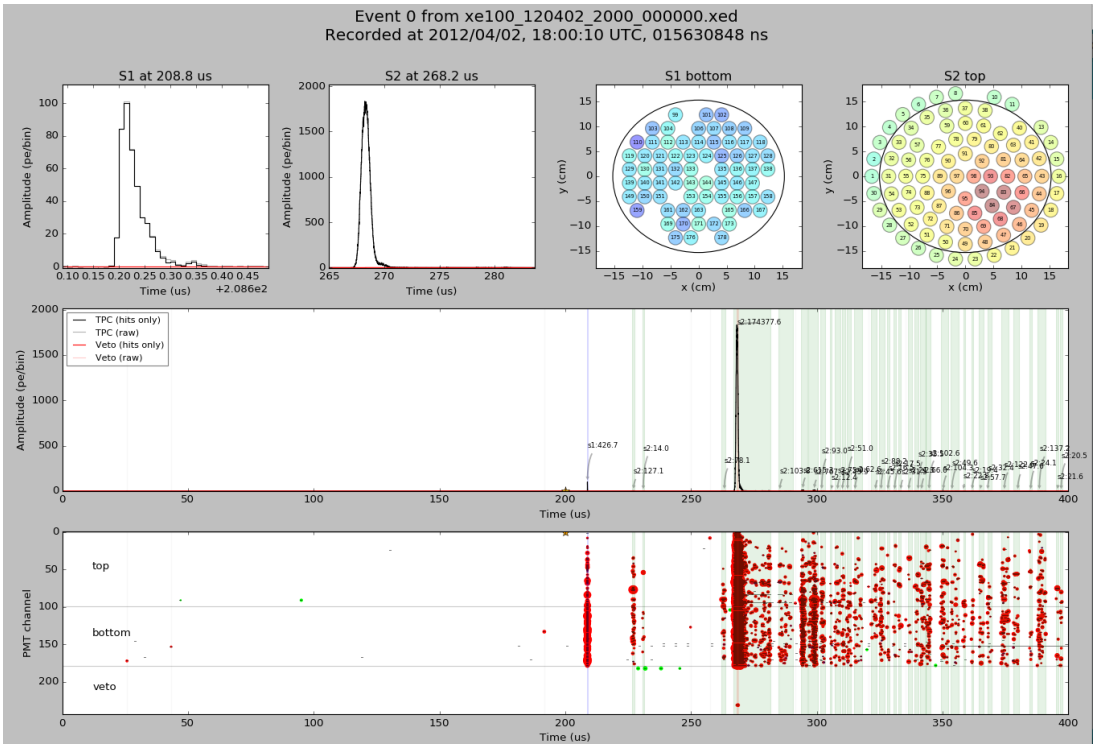

Figure 2: The Processor for Analyzing XENON (PAX) software for raw data processing

in Python, which allows easier and flexible "pythonic" access to the ROOT data, without the inconvenience of re-looping over the events each time a new plot needs to be created or adjusted. To extract and further reduce the data (make cuts), HAX lets researchers create "mini-trees"; small, tabular ROOT structures for Python analysis, which can be read directly into *pandas* DataFrame structures. In the following section we explain in more details how we implemented this computing model.

3. Beyond HEP-specific tools

The XENON scientific community consists of about 150 researchers, with no dedicated software-expertise manpower. The new computing model was prototyped to address some of the issues that small-to-medium-sized particle physics experiments face: limited resources and manpower to develop, maintain, and support the necessary software libraries. The bigger LHC experiments, such as ATLAS and CMS, define the standards of particle physics computing for more than two decades, relying heavily on CERN's ROOT framework and file format for physics analysis. The observation that the HEP computing community is becoming less isolated, is already reported [5]. However, the process is rather slow, as the HEP community seems to be neither very interested nor ready to accept more mainstream generic solutions, or provide such solutions. There are exceptions to this, of course, but the list is rather short. Examples of projects with beyond-HEP applicability and impact are DIRAC [6], GEANT4 [7], and PanDA [8].

On the other hand, generic Big Data analytics tools are gaining a momentum in scientific data analysis, due to the (much) larger and dedicated community support behind these mature tools. Python has become the language of choice for high-level applications where fast prototyping and efficient development are important, while glueing together low-level libraries for performance-critical tasks. The processing software (including the Event data model) is developed fully

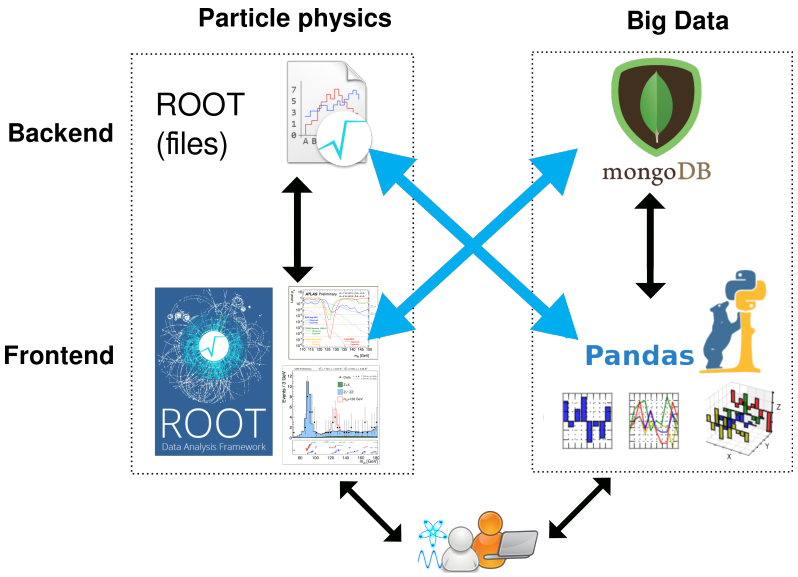

Figure 3: Data flows in Particle Physics and Big Data. Interfacing ROOT I/O in Python *pandas* will transitively allow interoperability between existing tools in the two ecosystems

in Python 3. We have found that ensuring correctness via unit testing, integration testing, documentation, and code sharing in Python and IPython notebooks has many advantages compared to our existing particle physics codes. We put significant efforts into following Open Source standards and procedures necessary to maintain coherent, sustainable, and easy-to-use software. In addition, relying on existing analytics code, instead of particle-physics specific code, greatly simplifies our work and allows us to focus on the physics.

However, within the XENON collaboration, there is a tension between these "novel" Big Data solutions, and the existing ones used in the wider HEP community. To ease the transition, our computing model should cater to both analysis paradigms, leaving the choice of using ROOT-specific C++ libraries, or alternatively, Python and its data analytics tools, as a front-end choice of developing physics algorithms. Hence, our goal is to harmonize these two ecosystems, shown on Fig 3, by organizing software and data such that we can work with the existing particle physics infrastructure, yet still use modern communal Big Data tools. Specifically, we interfaced Python *pandas* DataFrames to the ROOT format, which allows us to use off-the-shelf software libraries (e.g., NumPy, SciPy, Scikit-learn, Matplotlib) and lower the cost of developing and maintaining the XENON software stack. With pandas I/O to ROOT, we can also take advantage of tools such as the *odo* project [9] for data migration between storage systems (part of the PyData stack), and allow for conversions between pandas, HDF5, and MongoDB formats. In addition to helping us discover dark matter interactions, lowering this barrier helps shift the particle physics toward non-domain-specific codes.

3.1. Interfacing ROOT to Python pandas DataFrames

Besides the "social" reasons (hesitance to make a big radical leap) for keeping ROOT as the backend file format, there are valid technical ones as well: column stores, continuously improved techniques for pre-fetching, caching, and flushing of data buffers. Over the past years, the ROOT I/O team has significantly enhanced the I/O performance of ROOT [10]. However, as already stated, nowadays many "physics-specific" analysis algorithms are not that "physics-specific" in fact. Python has tools and libraries for efficient data structures and array manipulations, with a

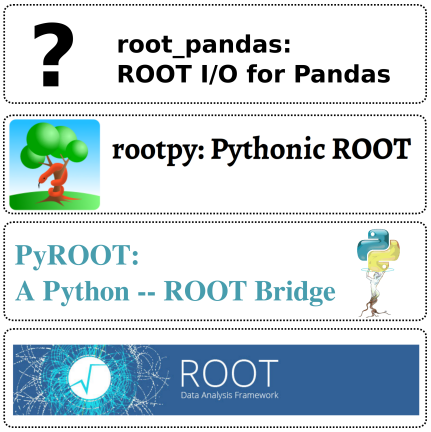

Figure 4: Interfacing ROOT to Python pandas DataFrames

steeper learning curve than the ROOT analysis framework. We further elaborate on the existing layers (Fig. 4) we used and adapted to fulfill the goal of interfacing ROOT to the Python pandas DataFrames.

The PyROOT [11] module is a bridge between Python and C. Despite its ROOT bindings, it is merely a mechanical translation of C++ function calls, and as such, programming is not really "pythonic", i.e., one can not fully leverage existing experience with Python; the native Python data structures are not used as expected, and these function calls do not adhere to the common Python idioms. Furthermore, Python 3 is not yet fully supported, and at the moment of writing, there is no official maintainer of PyROOT. Often both Python 2 and 3 versions are available on users' systems. However, the official binary releases of ROOT have the PyROOT module built with only Python 2 support, so users must resort to compiling ROOT from source. Steering the compilation process to look for the correct Python executable and libraries is cumbersome, and requires setting up environment variables and symlinks. One of our goals was to significantly simplify this process for the collaboration. In Section 4 we elaborate our approach to achieving this goal. In addition, there were blocking Python 3 issues with memory leaks [12] when using array fields, which we had to resolve by patching PyROOT.

The rootpy [13] project is a community-driven effort aiming to bring ROOT closer to the Python arena, on top of the existing PyROOT bindings. In addition to its main capability of creating ROOT tree data models purely in Python (Listing 1) and easier navigation through ROOT files, rootpy provides an interface with matplotlib, a popular Python publication-quality plotting library. It can further redirect ROOT error messages through Python's logging system, turning them into Python exceptions. The related root_numpy [14] library is used for efficient conversions between ROOT TTrees and structured NumPy arrays. One can then take advantage of the statistical and numerical packages that Python offers. The root_pandas [15] package is a convenience wrapper built around the root_numpy library. It allows to easily load and store pandas DataFrames in the ROOT format.

One essential problem that we stumbled upon, with both rootpy and root_pandas, is that creating trees with branches of user-defined types (and arrays/lists thereof) still requires the necessary "glue" C++ code. We currently autogenerate the C++ classes inheriting from TObject, based on introspection of our existing Python Event data model (sketched in Listing 2). Without it, these tools cannot handle nested (json-like) structures, such as the ones PAX uses. Storing non-rectangular data into Python DataFrames is possible, but it results in object structures that are not easily queried in a manner that flat data can be. This is why we created another layer, the HAX analysis tool, which essentially uses root_numpy under the hood, to load extracted

```python
from rootpy.tree import Tree, TreeModel
from rootpy.tree import FloatCol, IntCol
from rootpy.tree import FloatArrayCol, CharCol
from rootpy.io import root_open
from random import gauss, choice

f = root_open("test.root", "recreate")
# define the model
class Event(TreeModel):
    s = CharCol()
    x = FloatCol(default = 'nan')
    y = IntCol(default = 0)
    f = FloatArrayCol(5)
tree = Tree("test", model=Event)
# fill the tree
for i in range(5):
    tree.s = ord(choice(ascii_letters))
    tree.x = gauss(.5, 1.)
    tree.y = i
    for j in range(5):
        tree.f[j] = gauss(-2, 5)
    tree.fill()
tree.write()
f.close()
```

Listing 1: Creating tree models with rootpy

```python
class ReconstructedPosition(object):
    x = float('nan')
    y = float('nan')
    ....

class Peak(object):
    area = 0.0
    detector = 'ptc'
    ...
    rec_positions = list(ReconstructedPosition)

class Hit(object):
    channel = 0
    center = 0.0
    ...
class Event(object):
    event_number = 0
    dataset_name = 'Unknown'
    ...
    peaks = list(Peak)
    hits = np.array([], dtype=Hit.get_dtype())
```

Listing 2: The pax Event model

ROOT "mini-trees" data into pandas DataFrames. We were not able to find a suitable mature library or a file format that deals with this type of "ragged" data in a more elegant and flexible manner for users.

4. ROOT in the Anaconda Cloud

The XENON data processing software makes extensive use of scientific data analysis libraries (such as the SciPy stack). To make it straightforward for the collaboration users to install the software in user-space, we strongly recommend the Open Source Anaconda Python distribution [16], which includes many Python scientific modules bundled in a single-click installation. This distribution installs all the dependencies cleanly in a single directory (known as *environment*), and does not interfere with other Python installations present on a system. Anaconda comes with a binary packaging, dependency, and environment management system named Conda [17]. It is the main interface for managing multiple installations of various binary packages in separate environments. Conda environments are not Python-specific, and offer additional advantages compared to virtualenv or pip. We aimed to make it easy to re-create consistent environments and be able to reproduce the work of collaboration members, so we adopted Anaconda as a long-term solution in our workflow.

As already mentioned, one of the bigger barriers users typically faced was getting ROOT installed in user-space. Rather than relying on users compiling it with the specific Python 3 bindings and resolving all dependencies themselves, our goal was to provide a ROOT binary distribution that would be as easy to install as `conda install root={version}`. Building a cross-platform binary conda package involves creating a *recipe*, containing metadata on the source code, the build and run dependencies, as well as the scripts needed to build the package. In many cases creating a recipe is straightforward; however, making a "as portable as possible" ROOT installation is somewhat involved, due to its dynamic dependencies on `GCC` and libstdc++.

Linux distributions typically ship a particular version of `GCC` and `glibc`. Our XENON users need to be able to install and run ROOT on a variety of OS versions, ranging from reasonably old to new. Given that ROOT 6 requires `GCC>=4.8` to use some of the latest `C++` features, while

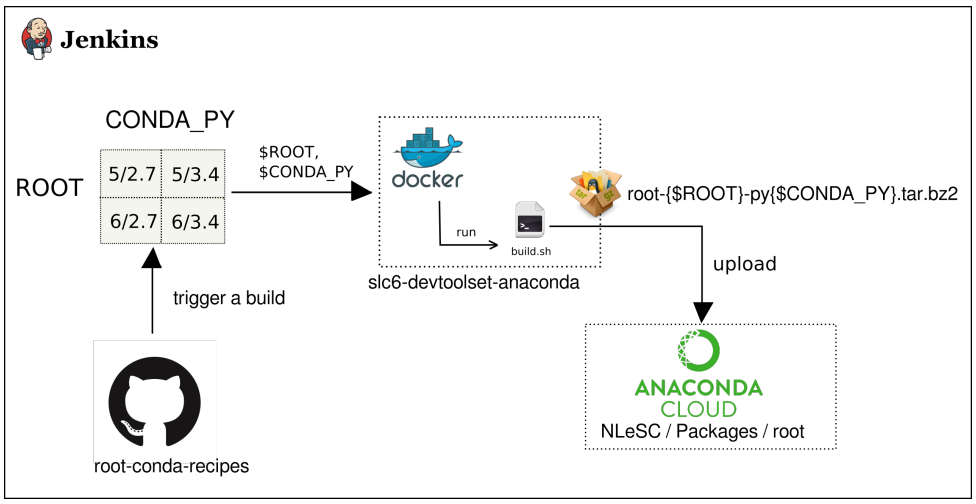

Figure 5: Jenkins CI setup for building ROOT binaries and uploading them to Anaconda Cloud

we aimed to create a single binary distribution that would work out-of-the-box on older systems, we decided to ship `GCC 4.8` as a ROOT runtime dependency package in conda. To maintain the ABI (binary) compatibility, and go "as low as possible" with `glibc`, the solution we found is a hybrid one: provide a `GCC 4.8` conda binary that is compiled against an older Linux kernel. To meet this challenge, SLC 6 offers a good compromise: the Developer Toolset (v2) from CERN provides a newer complier and binutils, while the kernel remains untouched. This way we can provide a `GCC/libstdc++` that supports the latest `C++` features, but is still compatible with the oldest `libc` and kernel we can support. These technical choices allowed us to produce ROOT (version 5 and 6) conda binaries which have been sucesfully tested on Ubuntu (11.10, 12.04, 14.04, 15.04) and SLC (6.5 and 7). The ROOT binaries (with Python bindings for 2.7 and 3.4) and dependencies are available from the Anaconda Cloud, a service for hosting (free public) conda packages. This also allowed us to do automated integration testing of the XENON data processing software within the Travis CI. All recipes are also publicly available [18], along with a documentation on their usage.

4.1. Continuous integration

Our goal was to have the ROOT (and dependencies) conda packages be automatically rebuilt, tested and uploaded to the Anaconda Cloud, triggered by pushing changes to the GitHub recipes repository. There are two mature options for this: Travis and Jenkins CI. Travis CI is the most convenient way to automate the builds for both Linux and OSX platforms, and it is well integrated as a free service with GitHub. Unfortunately, the current build time restrictions do not allow us to build ROOT. Depending on the number of features which are enabled, building and uploading ROOT 6 binaries can take up to (more than) 120 minutes, even with the most bare-bones configuration.

A public Travis CI setup is used for testing the sanity of existing ROOT conda binaries. Once the build-time restriction is lifted or relaxed, switching the building process on, is rather trivial. For the moment, we rely on an externally-hosted Jenkins machine where we have more resources. The setup is shown in Fig. 5. Each packaged binary of the build-matrix is created using a Docker image equipped with CERN's Developer Toolset (v2), cmake, and conda/Anaconda installed. `CONDA_PY` and `ROOT` are environment variables (configurable in Jenkins), and are passed to the *build.sh* script, which contains the `conda build` and `upload` commands.

5. Evaluation

...some XENON1T workshop trainings? Jeff's anaconda Nikhef setup at Stoomboot? centralized installation? xecluster? Pandas examples from XENON1T/hax?

6. Conclusions and future work

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

References

[1] Aprile E *et al.* 2012 *Astroparticle Physics* **35** 573–590 ISSN 0927-6505 URL http://www.sciencedirect.com/science/article/pii/S0927650512000059

[2] Aprile E, Alfonsi M, Arisaka K, Arneodo F, Balan C, Baudis L, Behrens A, Beltrame P, Bokeloh K, Brown E *et al.* 2014 *Astroparticle Physics* **54** 11–24

[3] Aprile E 2013 The XENON1T dark matter search experiment *Sources and Detection of Dark Matter and Dark Energy in the Universe* (Springer) pp 93–96

[4] Solem A 2014 Celery: Distributed Task Queue

[5] Smirnova O, Brun R, Cailliau R, Carminati F and Elmer P 2014 *Journal of Physics: Conference Series* **513** 052033 URL http://stacks.iop.org/1742-6596/513/i=5/a=052033

[6] Tsaregorodtsev A, Bargiotti M, Brook N, Ramo A C, Castellani G, Charpentier P, Cioffi C, Closier J, Diaz R G, Kuznetsov G, Li Y Y, Nandakumar R, Paterson S, Santinelli R, Smith A C, Miguelez M S and Jimenez S G 2008 *Journal of Physics: Conference Series* **119** 062048 URL http://stacks.iop.org/1742-6596/119/i=6/a=062048

[7] Agostinelli S, Allison J, Amako K a, Apostolakis J, Araujo H, Arce P, Asai M, Axen D, Banerjee S, Barrand G *et al.* 2003 *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **506** 250–303

[8] Borodin M, De K, Jha S, Golubkov D, Klimentov A, Maeno T, Nilsson P, Oleynik D, Panitkin S, Petrosyan A, Schovancova J, Vaniachine A and Wenaus T 2014 PanDA Beyond ATLAS : A Scalable Workload Management System For Data Intensive Science Tech. rep. ATLAS Collaboration, CERN URL https://cds.cern.ch/record/1670021

[9] Continuum Analytics Odo: Shapeshifting for your data http://odo.pydata.org

[10] Canal P, Bockelman B and Brun R 2011 *Journal of Physics: Conference Series* **331** 042005 URL http://stacks.iop.org/1742-6596/331/i=4/a=042005

[11] Lavrijsen W 2015 *Journal of Physics: Conference Series* **664** 062029 URL http://stacks.iop.org/1742-6596/664/i=6/a=062029

[12] CERN Central Issue Tracking Service https://sft.its.cern.ch/jira/browse/ROOT-7854

[13] Dawe N, Waller P, Friis E K, Deil C, schmitts, Turra R, Klukas J, Stevenson S, Buat Q, chrisboo, Alessandro, Kreczko L, Hegeman J, Verzetti M and Hollingsworth M 2015 rootpy: 0.7.0 URL http://dx.doi.org/10.5281/zenodo.18815

[14] Dawe N, Ongmongkolkul P, Deil C, Stark G, Waller P, Howard J and Babuschkin I 2015 root_numpy: 4.4.0 URL http://dx.doi.org/10.5281/zenodo.31192

[15] Babuschkin I, Remenska D, Nöthe M and Pearce A 2016 root_pandas: Initial release URL http://dx.doi.org/10.5281/zenodo.45464
[16] Continuum Analytics Anaconda: a free Python distribution for scientific Big Data analysis https://www.continuum.io/why-anaconda
[17] Continuum Analytics Conda: open source package and environment management system http://conda.pydata.org/
[18] Remenska D 2016 root-conda-recipes: Initial release URL http://dx.doi.org/10.5281/zenodo.47512