HEP analysis ecosystem workshop
Missing Pieces and Conclusions

Pere Mato, EP-SFT Group Meeting - 29th May 2017

A (typical?) HEP Analysis in 2027…

Graeme Stewart

- Primary analysis inputs are made centrally in the experiment production system from Run 4 LHC data
- I send a high level description of a selection in [MFL] to the grid to make a reduced sample for my sub-group
  - Catalogs for data discovery are used, but specifics are irrelevant to me
  - Task bookkeeping ensures I know what events went into my skim and how much data was run over
  - Data gets stored in my cloud storage area
- Describing my analysis workflow in declarative [MFL] I iterate over different selection criteria
  - Using some reduced sample to pre-test
  - This could be happening on my 128 core laptop with 1TB of XPoint memory, or via some notebook backed by a cluster

- MFL - My Favorite Language
- Two phases:
  - Production of reduced samples of selected events in the Cloud
  - Iteration over on ‘laptop’ or/and ‘analysis service’

A (typical?) HEP Analysis in 2027…

Graeme Stewart

- I realise my event selector needs better training
  - I skim off a selection of variables from my monte carlo in ROOT format
    - There are other formats, but I like this one as it's storage efficient
  - These are loaded up and connected to Keras++ via numpy arrays (one choice of many)
    - After using a backend that I don't even know what it is I have a trained DNN selector
- I rerun by selection (updated git tag!) with the new selector that ROOT has loaded from its native format
- Calibrations and enhanced object selections done here
  - Luminosity and other book keeping is automated
- Outputs are now small enough to download to a local machine
  - I still do my plots in ROOT, because it's the best for HEP data

- Automated handling of non-event data (lumi, bookkeeping)

Graeme Stewart

Key Elements

- Easy access to bookkeeping information and other metadata
  - Common support for this across experiment boundaries
- Declarative language (today we like python) used to describe tasks
  - Describe what, not how
- Analysis scales easily and transparently to different resources
  - Executor will marshall different components as needed
  - Integration with cloud/grid resources and special resources
  - Workflow engine is very robust - *user* does not need to worry about 1% failures
- Easy integration with other tools
  - Both in persistent formats and transient representations (like numpy and the zoo Jim showed)

This is a significant *evolution* of the current model, but keeping the model flexible will allow for the fact that:

- Not everything can fit into the one model anyway
- It should be possible (easy?) to make radical departures from the norm of one or more pieces - there may be a revolution and we just don't know it yet

- Most probably an evolution of the current models

Important Points

- **ROOT** continues as a foundational layer, and a connector with other technologies (thanks to its reflection system)
- **Python** should be a first class language (needs full support)
- HEP needs to **better leverage external tools** to its benefit
- **C++** will persist and still be in use in 2027 (high performance language, big code base legacy ~50 MLOC in HEP)
- Model of **successive stages of data reduction** is still the baseline model (finally analyzing a compact dataset interactively)
- **Functional, declarative programming**. Splitting the "*what I want*" from the "*how it's produced*," which avoids over-specifying incidental details in general
- **Automation** of workflows, pipelines (leveraging open source CI tools)

Important Points (2)

- **Modularity**. Clean packaging, easy installation of components via package managers. Containers to assemble working stacks
- **Sharing infrastructure** will be more and more important. Including sharing specialized facilities, services, GPU farms, TPU farms etc.
- **Memory resident analysis** for late stage analysis
  - Need to support both **local and remote analysis.** Syncing services essential
- **Metadata handling** (e.g. lumi) is an important element. Need for a common customizable solution
- Need to worry about **reproducibility and preservation**. Code is data.
- **Event throughput** is the definitive metric for measuring performance of workflows
- **Avoid re-implementation** in particular for machine learning tools

Specific ROOT Feedback

- PyROOT
  - Need (long-term) maintainer
  - Better pipes to/from NumPy and other ML packages
  - Build on top of CLING (standalone for non-HEP community)
- RooFit essential
- Minuit is important and unique (error treatment)
- Recipes for common package managers (spack, conda, homebrew, etc.)
- Modularization of components
  - Late installation of pythia, hdfs, xrootd, etc.
- Support for other persistency backends
  - Efficient conversions from/to transient TTrees