Software Management and Deployment for the Muon g-2 Experiment at Fermilab

James Stapleton
Muon g-2 Postdoc
2019-09-10
Muon g-2

observe ~160 spin precession periods

Momentum
Spin

DHMZ17
KNT18
BNL

(a_{\mu}^{SM} x 10^{-10}) - 11659000

3.7\sigma
Muon g-2

observe ~160 spin precession periods

Momentum
Spin
Muon g-2

observe ~160 spin precession periods

Momentum
Spin

G.W. Bennett et al.
Muon g-2 Software & Computing

• Nature of g-2 shapes computing resources
  – quasi-HEP experiment + NMR experiment
  – one precession analysis

• Medium-sized experiment
  – adopted many Fermilab-native tools
  – developers: wide variety of computing & physics backgrounds
  – most code written by a few distinct small groups

• I won’t cover everything here!

• Just my personal experience with:
  – CVMFS
  – code development & release for offline processing & analysis
  – interactive access to data in framework files
CernVM FileSystem (CVMFS)

- POSIX read-only user-space filesystem (FUSE)
- Optimized for distribution of program files!
  - low-latency, on-demand directory listing & single-file access
  - entirely HTTP
  - aggressive caching
  - easy parallelization for computing grid nodes
- Accessible at data centers worldwide ("already there" for FermiGrid and many other computing facilities)
- Lots of handy features
  - low-maintenance
  - easy to publish files, revisioned filesystem image with named tags
  - stability on client (no FUSE problems, always mounts & unmounts cleanly, handles network outages well)
  - distributed mirroring (provided setup of certain networks)
  - good configurable parameters for clients, but defaults are well-chosen!
CVMFS for Muon g-2

- We have CVMFS mounted on our group VMs, on grid nodes, and in the occasional personal VM or container (all SLF6, for now)
  - distributes in-house software as well as externals to cover dependencies
- Externals: requirements like build tools, Python interpreter, ROOT, Geant4, etc
- Muon g-2 CVMFS share:
  - two people authorized to publish to Stratum-0
  - build tool can [de]select packages from CVMFS installation & set up an area for local build to replace them
  - periodic releases cut from several (17) git repositories
- Online & Offline: supplies code for offline running, but also hosts installations of online code (DAQ, quality monitoring) for offline compile-time & run-time dependencies
- Our user experience with CVMFS has been excellent
- Use with a VM: great for times when WiFi is far away...
CVMFS for Muon g-2

- We have CVMFS mounted on our group VMs, on grid nodes, and in the occasional personal VM or container (all SLF6, for now)
  - distributes in-house software as well as externals to cover dependencies
- Externals: requirements like build tools, Python interpreter, ROOT, Geant4, etc
- Muon g-2 CVMFS share:
  - two people authorized to publish to Stratum-0
  - build tool can [de]select packages from CVMFS installation & set up an area for local build to replace them
  - periodic releases cut from several (17) git repositories
- Online & Offline: supplies code for offline running, but also hosts installations of online code (DAQ, quality monitoring) for offline compile-time & run-time dependencies
- Our user experience with CVMFS has been excellent
- Use with a VM: great for times when WiFi is far away...
‘Off-Label’ Use

- Serve as a text file ‘database’ (calibration constants, etc)
  - NOT exactly recommended for calibration constants (but works great if you need it to!)
- Online/DAQ operations: very convenient for quick software deployment
  - not possible to track clients, which matters during breaking upgrades (lots of people forgot they were getting libXYZEtc from CVMFS)
- Lots of other convenient corner cases not targeting Offline purposes
## CVMFS Strengths & Limitations

### Strengths
- Excellent for program files
- Absurdly stable
- Bandwidth efficiency
- Easy to scale on grid (HTTP caching)
- No concurrency issues
- Easy client installation & configuration (c.f. NFS, Samba)
- Unexpected shutdown precipitates no issues due to journaling, etc

### Limitations
- Read-only, not easy to track clients*
- Scalability *like Muon g-2 has seen* requires setup of Stratum-1 network & HTTP grid caching, some coordination with remote processing sites
- Can be finicky about file permissions
- Does not like >200k entries in the same volume*

* More design choice than limitation
Code Development & Release

- We have 20-25 git repositories with different purposes.
  - official ‘Offline Releases’ include 17 interdependent repos
  - 10-20 active developers (<10 very active)

- History has driven evolution of code
  - Summer 2016: SLAC testbeam run (calorimeter)  Releases not critical
  - Summer 2017: Ring/Beam Commissioning
  - Winter-Spring 2018: Physics Run 1
  - Spring-Summer 2019: Physics Run 2
  - Summer 2018-present: Run 1 Analysis
  - Near future (post Run 1 Analysis)  Release every ~month

Run 2 release every two weeks
Run 1 release as-needed
Code Development

Physics Run 1 Analysis Effort

Physics Run 2

June/July 2017: Beam/Ring commissioning
Code Repositories

NOT shown: ~half a dozen ‘non-release’ packages
Flexible & Responsive Release Management

- Adapt! (primary customer changes over time, as well as core product)
- Need intuition for the growth & stability of the codebase
- Regular or irregular releases? Depends on experiment’s growth phase!
- Flexible coding conventions, development cycle recommendations
- CRITICAL for good administration of g-2 software:
  - contact with developers (communication!)
  - solicit discussions of code evolution often, keep an eye on important tasks, respond to developers’ timelines & priorities
  - gauge relative strengths/weaknesses of developers in order to help them efficiently
  - keep a Linux expert on hand ; - )
**art**

- **art**: Fermilab’s event-processing framework, developed in-house
  - greatest asset: responsive developer team
- File format is custom specification on a ROOT file substrate: put Data Products (arbitrary objects with a system of labels) into a Run, SubRun, or Event
- Events processed through a Module Path, modules append Data Products to Event, SubRun, or Run
- Analyzer module base class requires implementation of `void analyze(art::Event & e)` (executes once for each event)
- Data Products accessed by Handle
  - requested from Event, SubRun, or Run via templated member `GetValidHandle<>()`
  - specify Module Label, Instance Name, and Process ID
  - specify C++ type of Data Product in templated call to `GetValidHandle<>()`
Data Inspection

• Accessing file content is nontrivial

• The Only Native (Event-based) Interface to Content: create a new framework module and run it!
  – implement a new module, build & link to framework stuff, create new config file, load/run framework executable, etc
  – ...and the event-processing software exists in an environment with lots of requirements (only specific OSs, exact compiler version enforced, etc)

• Non-framework access is possible through TTree/TBrowser, but this has limitations (i.e. unintended use)

• Problem: no spontaneous & interactive route to inspecting contents of data files
Refactor: *art*, *canvas*, *gallery*

- *art* developers moved ‘Data Provenance’ code to the new dependency *canvas*
- new package *gallery* provides similar interface:
  - native ‘event loop’
  - fetch Data Products by handle
  - good for prototyping *art* framework code
  - (read-only)
- available via gcc, ROOT macro, or Python
- *gallery* via Python suggests *interactive inspection of data*
heist!

• Bare Python interface is clunky, so heist Python module wraps gallery interface generated by PyROOT

• A heist script is analogous to a single module directly fed art Events from file
  – event loop, data product handles, dereference `art::Ptr()` (smart pointer to another data product), easy to skip events with no matching entries
  – ...plus the introspection/reflection awesomeness of Python!

• Includes some extra useful tools:
  – list Data Products in a file, search by simple match, and a magic `ls()` function to describe Data Products (and lists of them)

• I use heist nearly exclusively, and we use it in the analysis group I lead

• Others in Muon g-2 have started to pick it up

• Almost ready to advertise to other experiments
and it works in a Jupyter notebook!
Illinois Ratio Method Analysis (IRMA)

- Muon spin precession analysis group
- Fermilab + University of Illinois, Urbana-Champaign
- Named for the ‘Ratio Method’ (but we are not restricted to that)
- Target: analysis of Run 2 data
Final Notes

- Muon g-2 are happy CVMFS customers!
  - does exactly what is advertised, no headaches
- Zen of computing: it’s really about people (not computers)
  - true for our software, and true for Fermilab’s support (art, etc)
- Scientific computing designs make assumptions about data (content, format/type, sizes) but should also assume
  - unexpected variation in data
  - a human will need to inspect the data at various points