Root-Based Analysis in ATLAS

Nils Krumnack (Iowa State University)

Introduction

- pre-run I: all analysis is to be done in Athena
  - Athena: official framework used for production and trigger
  - working with its own data format: AOD
  - root-only code only to be used for making plots
    - no corrections, calibrations, etc. in this step
  - big focus on uniformity, reusability and reproducibility

- then came collision data:
  - few people actually followed the official model
  - most dumped their data into a "flat" n-tuple instead
  - most analysis tools applied on n-tuples

- had to develop new framework as data came in:
  - no grand concept, address problems by priority
  - very limited manpower for development (1-3 FTEs)
  - needed to integrate with code users wrote so far

RootCore Build System

- first thing we needed was a build system:
  - every tool came with its own build system
  - some users spend two weeks(!) on getting all tools to compile

- first approach very simple:
  - each makefile included a Makefile-common
  - in turn included Makefile.arch from root package
    - extremely useful file
    - provided instant portability
  - some extra shell scripts for core functionality

- extremely crucial part:
  - provided conversion for all common tools
  - open-ended offer to convert user packages as well
  - strong support to all users
- very successful: quickly became widely used, very positive feedback

RootCore Evolution

- RootCore evolved a lot since the early days
  - mostly driven by user requests
  - or to address recurring user problems

- supports package management
  - user provides text files with package lists
  - RootCore handles the SVN operations
- support for grid and batch submission
  - generates a self-contained directory with all packages
  - includes scripts for setup on worker nodes
- initialization scripts for interactive root
  - loads all libraries and performs any special setting needed
- support for various external packages
  - downloads and compiles source for the user
  - makes them portable and easy to use

Data Formats

- early on ATLAS developed a "flat" n-tuple dumper for Athena:
  - i.e. one branch for each variable
  - data format called D3PD
  - works well for analysis, which often only reads a subset of data

- later on joined by a corresponding reader:
  - performed read-on-access optimization
  - rebuild objects like electrons from individual variables
- greatly simplified/optimized reading D3PDs

EventLoop Job Management

- needed a job management system that:
  - provided better job handling on the grid
  - made it easier for users to use PROOF
  - allowed to switch easily between local, batch and grid running
- basic design is fairly standard:
  - user writes code as algorithms (similar to a TSelector)
  - different drivers handle running in different locations

- originally assumed this could be a backend for other frameworks:
  - users can easily use their own algorithm base class
  - submission/configuration easily wrappable as well
  - nobody ever used this flexibility
- successful with new users, but not for existing projects:
  - migration typically involves some code re-writes
  - legacy code often already does what users need

EventLoop: Algorithm Design

- EventLoop uses streaming algorithms:
  - user configures algorithm in his submission script
  - then gets streamed via root-i/o to the worker nodes
  - sidesteps any need for configuration by EL
- really regret this design decision:
  - causes majority of user problems with EL
  - many users have problems writing streamable objects
    - in root 5 many header files were not parseable

- had briefly flirted with using TSelector instead, but:
  - having own class provides greater flexibility
    - extended it repeatedly
  - harder to simplify PROOF usage if we use the same interface
  - allows syntactic similarity to Athena

Further EventLoop Design

- provides no data model or white board service:
  - most users use a single algorithm anyways
    - i.e. they don't need communication mechanisms
  - possible to plug in data model as extra algorithms
  - allowed EventLoop to remain unchanged as data model evolved

- instead of submission id, using a submission directory:
  - i.e. a unique directory for each job submission
  - holds output data, configuration and temporary data
  - allows drivers to store any extra data they need
  - no need for a central job database

EventLoop & PROOF

- PROOF-lite support fairly straightforward
- main issue: no way to run code on empty files
  - needed for in-file meta-data stored in each file
  - workarounds possible, but native support would be nice

- PROOF-farm support is a continual head-ache:
  - impossible to test without a PROOF farm
  - non-trivial to debug even with a PROOF farm
  - bad fit between ATLAS build system and PROOF setup system
- users do like the PROOF-farm though:
  - provides strong performance improvements

- a less integrated system would probably work better:
  - i.e. EventLoop schedules and starts the jobs
  - within each job create a PROOF client object
  - ask the PROOF client object which event to process next

SampleHandler

- an ATLAS analysis may require over a hundred datasets
  - mostly different background samples
  - also need to track sample meta-data like luminosity
  - tedious to do manually for large number of datasets
- manages list of files and meta-data per dataset
  - can also pre-stage files as needed (rarely used feature)
  - can manage local and grid datasets
- contains various data discovery methods
  - typically written by expert when users request them
  - allows to organize the input data in various ways

- success is somewhat mixed:
  - lots of people use the basic functionality for EventLoop
  - few people use (all) the advanced features that would help them
  - mostly because it is hard to communicate all that SH can do

SampleHandler II

- basic implementation very simple
  - nothing difficult about a file list
  - some extra features for pre-staging, remote access, etc.
- only snafu is meta-data:
  - ideally want something like std::map<std::string,boost::any>
  - but root can't stream that
  - instead use TList, wrapping everything into TObjects
  - seems to be a recurring problem in several tools

- main advantage is to have single format for per-dataset data
  - can be filled from various sources: text files, central database, etc.
  - decouples tools from input source

MultiDraw Plot Making

- in principle TTree::Draw is pretty nice:
  - it can be taught to anyone in minutes
  - its interface is very intuitive
  - it is highly optimized
- however: doesn't scale well

- MultiDraw solves that:
  - wraps TTreeFormula into an EventLoop algorithm, e.g.

```
new AlgHist (new TH1F (...), "el_n")
```

  - can schedule as many algorithms as needed
  - can run on any batch system via EventLoop
  - can run in addition to any "regular" algorithm
- don't know how many people actually use it:
  - mostly because it just works…
  - people who do use it seem pretty happy with it though

Transition To Run 2

- the long shutdown gave us chance for reorganization:
  - harmonized tools between Athena and root-only analysis
  - addressed some general issues

- merged file formats between Athena and root-only analysis
  - common interface classes in both environments
  - greatly simplifies writing tools serving both
  - reduces waste/incompatibilities from multiple formats

- centrally provide precompiled analysis releases
  - contains all common packages typically needed
  - saves users the trouble of compiling them
  - allows greater central control over packages used
- required two major rewrites of RootCore:
  - first to support the releases themselves
  - then rewrite it in python for speed…

New Tool Interface

- provide a new base class for analysis tools:
  - i.e. tools that provide corrections, selections, etc.
  - derives from different classes based on environment
  - allows the same implementation to work in both environments

- nifty new mechanism for systematics:
  - standard format/interface for systematic variations
  - allows multi-sigma variations
  - allows applying multiple systematics at once
- greatly simplifies systematics handling
  - can just loop over all tools to set the systematic
    - no tool specific code needed
    - correlations automatically handled
    - can happen before processing the event
  - allows simple loop of systematics per event

Analysis Frameworks

- during run I many groups provided analysis frameworks
- at their core they all do mostly the same
  - i.e. apply all the standard tools for the user
  - provide collections of fully corrected good objects to the user
  - provide an overall event weight to the user
  - evaluate all systematics in a standardized manner

- in general very useful:
  - simplifies life for the user
  - harmonizes work within groups
  - reduces potential for mistakes
- numerous problems as well:
  - often hard to understand what they do exactly
  - differences can make collaboration across groups impossible
  - not every group had a framework, leaving some users stranded
  - doesn't always integrate with other ATLAS software

QuickAna Tool Scheduler

- QuickAna is an attempt at a common analysis framework
  - runs all analysis tools for the user
  - provides final analysis objects to the user
  - support for users from all high-p_T physics groups
  - implements object definitions from various groups
  - make use of existing ATLAS software where possible
- try hard to avoid "black box" complaints
  - all code that makes physics decisions is separated from the rest
  - written in simple and straightforward C++
  - following closely the actual physics logic
  - should be understandable without looking at infrastructure code

QuickAna Object Definitions

- tools are grouped into object definitions
  - e.g. all electron tools form the electron definition

- multiple possible configurations per object type
  - the user chooses object definitions by name
    - e.g. "tight" electrons, "loose" muons, etc.
  - user can also choose multiple object definitions at once
    - can store multiple selections on same object
  - typically no extra configuration of individual tools needed

- a number of advantages:
  - easy to teach to newcomers
  - configuration matches physics presentations more closely
  - insulates users from changes in tools
- introduces an additional layer to configuration
  - translates physics configuration to tool configuration

QuickAna Optimized Running

- systematics evaluation is fairly simple:
  - apply systematics setting
  - run analysis code
  - change systematics setting
  - repeat dozens of times per event
- simple, robust, and wasteful:
  - systematic setting typically affects just one tool
  - rerunning other tools wastes CPU
  - storing their output wastes disk space too

- for optimization all tools report:
  - their inputs, outputs, and directly affecting systematics
- only re-run a tool if the systematic affects it or its inputs
  - for other tools use output from no-systematics evaluation
- saves about factor 2-3 in terms of CPU

Some Personal Lessons

- if you give users a choice between all or nothing sometimes they will choose nothing

- make life easy for newcomers
  - they outnumber the experts
  - experts can often handle the complications better
- support multiple ways of doing the same thing
  - hard to know which way is best beforehand
  - different users have different tastes/needs

- try to implement user feature requests
  - they typically know better than you what they need
  - but don't hesitate to do it your way
  - some of my best features were based on feature requests
- do implement features only 5% of users want:
  - a dozen such features is half your user base

Summary & Outlook

- during run I ATLAS developed a suite of analysis tools
  - mostly because users weren't using the main framework
  - forming a simple analysis frameworks by now

- used the long shut-down to reengineer our tools
  - made the two environments more similar
  - made it easier to switch between the two environments

- continuing to integrate the tools even more
  - aiming to have a single framework eventually

- trying to support work further down the analysis chain as well
  - i.e. applying all the corrections, etc.
  - simplify standard tasks like plotting, data management, etc.