Common CP Algorithms at ATLAS

Nils Krumnack (Iowa State University)
on behalf of the ATLAS computing activity

Introduction

- ATLAS analyzers need to apply a lot of code on top of input files
  - e.g. final calibrations, selections, scale factors
  - developed separately by respective domain experts
  - commonly called "CP recommendations"

- ATLAS has infrastructure for the user to apply them
  - allows to treat recommendations (mostly) as a black box
  - allows to get users started very quickly
  - allows easy rollout of new recommendations
- currently done via one of several "analysis frameworks"
  - each maintained by their respective user communities

- goal: a single analysis framework for all of ATLAS
  - further harmonization for ATLAS analysis
  - reduced maintenance effort
  - improved user experience

CP Tools at ATLAS

- a CP tool is an ATLAS "component" class for applying central analysis recommendations
  - performs calculations belonging to recommendation
  - configurable from python or C++
  - sharable between analysis, production and online code

- tools implement a tool specific C++ interface
  - member functions specific to what the tool does
  - inputs are EDM objects ← very stable interface

- CP tools also implement a common interface for systematics
  - allows to query list of tool systematics
  - allow quick changes between systematics
  - chosen systematic used for all subsequent calls

CP Tool Successes

- easy to implement and deliver recommendations
  - can customize tool interface for task at hand
  - using EDM objects in interfaces keeps interfaces stable
  - easy distribution via ATLAS software releases
  - built-in mechanism for distributing calibration files

- recommendations fairly straightforward to use
  - can (normally) treat the implementation as black box
  - (most) CP tools usable with 2-5 lines of code
  - can set configuration options on each tool as needed

- CP tools can be shared between analysis, production and online
- CP tool interfaces can hide very complex implementations

CP Tool Problems

- building an analysis from CP tools often non-trivial:
  - can involve using dozens of tools
  - each tool needs some custom code to call it
  - configuration needs to be consistent across tools
  - various subtleties and pitfalls
  - applying them consistently between analyses difficult

- numerous analysis framework evolved
  - take care of applying all CP tools
  - hide a lot of the technical details
  - provide extra functionality, most commonly n-tuple making

- numerous analysis frameworks in ATLAS these days
  - duplication of development/maintenance efforts
  - reproducibility between frameworks can be a problem
- want a single framework for everyone to use

Challenges for Framework

- need high degree of customizability for unified framework
  - single framework needs to cover all ATLAS users
  - need ability to select which object types to use
    - allow multiple copies (with different settings)
  - allow writing out both tight and loose selection for an object
  - support both main analyses and special studies

- separate default configuration from user configuration
  - domain experts provide/maintain the default configuration
  - users select configuration they want
  - users can override (most) settings as needed

- need efficient systematics handling
  - ATLAS analysis can have well over 100 systematics
  - want to minimize work to be done
  - want detailed bookkeeping of all systematics

CP Algorithms

- first challenge: CP tools are not "schedulable"
  - each tool has a custom C++ interface
  - requires custom C++ wrapper per tool
  - harder to add/remove tools based on configuration

- utilize concept of ATLAS algorithms:
  - single common interface, called once per event
  - input/output via a shared whiteboard
  - easy to setup "sequence" of algorithms in configuration
  - can add/drop/repeat algorithms as needed
  - concept already well established in reconstruction/online

- wrap CP tools in "CP algorithms":
  - one CP tool per CP algorithm
  - systematics loop internal to each algorithm
  - configuration creates full sequences for each object type

Systematics Handling

- want to run each tool only for minimal set of systematics
  - systematics directly affecting the tool
  - systematics affecting the tool's input
- needs full data dependency tracking
  - done at variable level, not object level

- access all inputs/outputs via "systematics data handles"
  - one data handle for each accessed object/variable
  - declares list of inputs/outputs for dependency tracking
  - allows access to data for current systematics
  - encapsulates all systematics handling code
  - code structurally very similar to code without systematics

- originally did systematics tracking during configuration
  - switched to post-configuration initialization
  - simplifies both configuration and dependency tracking

N-Tuple Output

- main use of analysis frameworks in Run 1-3:
  - produce "flat" n-tuple from centrally produced files

- traditional approach: separate tree for each systematic
  - simple to write out: global loop over all systematics
  - easy to analyze: global loop over all systematics

- CP algorithm: single tree, separate branch(es) for each systematic
  - utilize per-variable systematics tracking
  - know exactly which variable is affected by which systematic
  - extra bookkeeping during analyze

- branch-per-systematic much more space efficient
  - about 1-2 orders of magnitude
  - rather significant: can mean 10s of TB vs <1TB

Configuration

- original design: produce one sequence for each object type
  - sequence maker code contains actual physics configuration
  - maintained by respective domain experts
  - user can override settings on each tool/algorithm

- works well for individual sequences
  - encapsulates/hides many implementation details
  - well defined inputs/outputs

- composition of sequences more tricky:
  - need to track extra information besides sequence
    - e.g. selection flags created, operations applied, inputs used
    - affects downstream configuration (possibly also upstream)
  - need to manage temporaries created in whiteboard
  - need to eliminate duplicate operations

New Configuration

- new approach: build configuration from individual "blocks"
  - blocks are python objects that generate sequences
    → alg. sequence now produced after user configuration
  - blocks define their own options for the user
  - blocks responsible for interfacing with each other

- blocks communicate via central configuration accumulator
  - avoids blocks directly interacting with each other
  - two step process → allows passing information upstream (mostly used for managing temporaries)

- also working on a text-based configuration
  - assemble blocks from a yaml configuration file
  - goal: more abstract, physics-oriented configuration
  - specify what you want, not how to get it

Rollout at ATLAS

- rollout at ATLAS has been slow
  - fairly high effort to switch analysis frameworks
    - changes to configuration and n-tuple formats
  - benefits of single framework more abstract
  - existing analysis frameworks "good enough" for most users
  - most people agree to switch at some point

- used in central production of new PHYSLITE format
  - contains pre-calibrated objects for simpler/faster analysis
  - see Jana Schaarschmidt’s talk

- starting to see more active migration efforts lately
  - demonstrated size benefits of CP algorithms n-tuple
  - new configuration now available
  - more central role in beginner’s tutorial
  - first analysis framework developers getting involved

Summary & Conclusions

- ATLAS provides central recommendations to analysis users
  - CP tool mechanism well-established for years
  - insulates users from most implementation details

- integration of CP tools via analysis frameworks
  - removes most remaining complexities
  - allows users to get started very quickly
  - multiple analysis frameworks currently in use

- presented a unified analysis framework for all of ATLAS
  - being rolled out to the user community
  - used as foundation for further Run 3/4 software (see following talks)

backup slides

A Common Framework

- goal: have a common analysis framework for all of ATLAS
  - provide better physics harmonization
  - provide more flexibility to users
  - employ best practices in implementation
  - reduce duplication of development/maintenance efforts
  - supplement or replace existing frameworks

- no obvious framework to pick as the common one:
  - most popular framework had 15-25% of users
  - functionality not always good match to what we wanted
  - generally would need rework of implementation

- started a new analysis framework from scratch:
  - import best ideas from the different frameworks
  - most work went into central infrastructure developments
  - main challenge was clean/efficient systematics handling

Systematics Storage

- ATLAS EDM has shallow copy feature
  - make efficient copies of existing object containers
  - reuse all object variables that are not modified
  - adding/overwriting variables on copy doesn’t change original
- easy to do systematics with shallow copies
  - algorithms that add new systematics make new copies
  - data handles can create or lookup correct copies
  - works well for simple linear setups
- doesn’t work well for complex setups:
  - algorithms don’t use all object variables from their inputs
  - per-object systematics tracking picks up superfluous systematics
  - switched to per-variable systematics tracking instead
  - give different names to variables based on systematics